# Memory and attention in deep learning

by

**Hung Thai Le**

BSc. (Honours)

Submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

Deakin University

*August 2019*

**DEAKIN UNIVERSITY**
**ACCESS TO THESIS - A**
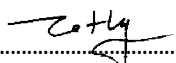
I am the author of the thesis entitled

**Memory and attention in deep learning**

submitted for the degree of **Doctor of Philosophy**

This thesis may be made available for consultation, loan and limited copying in accordance with the Copyright Act 1968.

*'I certify that I am the student named below and that the information provided in the form is correct'*

Full Name: ...............**HUNG THAI LE**............................................................................
(Please Print)

Signed: ....................................................................................................................

Date: ...............................09/02/2020.........................................................................

**DEAKIN UNIVERSITY**
**CANDIDATE DECLARATION**

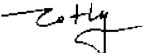I certify the following about the thesis entitled (10 word maximum)

**Memory and attention in deep learning**

submitted for the degree of **Doctor of Philosophy**

a.   I am the creator of all or part of the whole work(s) (including content and layout) and that where reference is made to the work of others, due acknowledgment is given.

b.   The work(s) are not in any way a violation or infringement of any copyright, trademark, patent, or other rights whatsoever of any person.

c.   That if the work(s) have been commissioned, sponsored or supported by any organisation, I have fulfilled all of the obligations required by such contract or agreement.

d.   That any material in the thesis which has been accepted for a degree or diploma by any university or institution is identified in the text.

e.   All research integrity requirements have been complied with.

*'I certify that I am the student named below and that the information provided in the form is correct'*

**Full Name:  Hung Thai Le**

**Signed:**

 **Date:    31/08/2019**

# Acknowledgements

# Contents

## 4    Memory Models for Multiple Processes                                          57

# List of Figures

# List of Tables

# List of Algorithms

# Abstract

Intelligence necessitates memory. Without memory, humans fail to perform various nontrivial tasks such as reading novels, playing games or solving maths. As the ultimate goal of machine learning is to derive intelligent systems that learn and act automatically just like human, memory construction for machine is inevitable.

Artificial neural networks model neurons and synapses in the brain by interconnecting computational units via weights, which is a typical class of machine learning algorithms that resembles memory structure. Their descendants with more complicated modeling techniques (a.k.a deep learning) have been successfully applied to many practical problems and demonstrated the importance of memory in the learning process of machinery systems.

Recent progresses on modeling memory in deep learning have revolved around external memory constructions, which are highly inspired by computational Turing models and biological neuronal systems. Attention mechanisms are derived to support acquisition and retention operations on the external memory. Despite the lack of theoretical foundations, these approaches have shown promises to help machinery systems reach a higher level of intelligence. The aim of this thesis is to advance the understanding on memory and attention in deep learning. Its contributions include: (i) presenting a collection of taxonomies for memory, (ii) constructing new memory-augmented neural networks (MANNs) that support multiple control and memory units, (iii) introducing variability via memory in sequential generative models, (iv) searching for optimal writing operations to maximise the memorisation capacity in slot-based memory networks, and (v) simulating the Universal Turing Machine via Neural Stored-program Memory–a new kind of external memory for neural networks.

The simplest form of MANNs consists of a neural controller operating on an external memory, which can encode/decode one stream of sequential data at a time. Our proposed model called Dual Controller Write-Protected Memory Augmented

Neural Network extends MANNs to using dual controllers executing the encoding and decoding process separately, which is essential in some healthcare applications. One notable feature of our model is the write-protected decoding for maintaining the stored information for long inference. To handle two streams of inputs, we propose a model named Dual Memory Neural Computer that consists of three controllers working with two external memory modules. These designs provide MANNs with more flexibility to process structural data types and thus expand the range of application for MANNs. In particular, we demonstrate that our architectures are effective for various healthcare tasks such as treatment recommendation and disease progression.

Learning generative models for sequential discrete data such as utterances in conversation is a challenging problem. Standard neural variational encoder-decoder networks often result in either trivial or digressive conversational responses. To tackle this problem, our second work presents a novel approach that models variability in stochastic sequential processes via external memory, namely Variational Memory Encoder-Decoder. By associating each read head of the memory with a mode in the mixture distribution governing the latent space, our model can capture the variability observed in natural conversations.

The third work aims to give a theoretical explanation on optimal memory operations. We realise that the scheme of regular writing in current MANN is suboptimal in memory utilisation and introduces computational redundancy. A theoretical bound on the amount of information stored in slot-based memory models is formulated and our goal is to search for optimal writing schemes that maximise the bound. The proposed solution named Uniform Writing is proved to be optimal under the assumption of equal contribution amongst timesteps. To balance between maximising memorisation and overwriting forgetting, we modify the original solution, resulting in a solution dubbed Cached Uniform Writing. The proposed solutions are empirically demonstrated to outperform other recurrent architectures, claiming the state-of-the-arts in various sequential tasks.

MANNs can be viewed as a neural realisation of Turing Machines and thus, can learn algorithms and other complex tasks. By leveraging neural network simulation of Turing Machines to neural architecture for Universal Turing Machines, we develop a new class of MANNs that uses Neural Stored-program Memory to store the weights of the controller, thereby following the stored-program principle in modern computer architectures. By validating the computational universality of the approach through

an extensive set of experiments, we have demonstrated that our models not only excel in classical algorithmic problems, but also have potential for compositional, continual, few-shot learning and question-answering tasks.

# Relevant Publications

Part of this thesis has been published or documented elsewhere. The details of these publications are as follows:

Chapter 4:

- Le, H., Tran, T., & Venkatesh, S. (2018). Dual control memory augmented neural networks for treatment recommendations. In Pacific-Asia Conference on Knowledge Discovery and Data Mining (pp. 273-284). Springer, Cham.

- Le, H., Tran, T., & Venkatesh, S. (2018). Dual memory neural computer for asynchronous two-view sequential learning. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 1637-1645). ACM.

Chapter 5:

- Le, H., Tran, T., Nguyen, T., & Venkatesh, S. (2018). Variational memory encoder-decoder. In Advances in Neural Information Processing Systems (pp. 1508-1518).

Chapter 6:

- Le, H., Tran, T., & Venkatesh, S. (2019). Learning to Remember More with Less Memorization. In International Conference on Learning Representations. 2019.

Chapter 7:

- Le, H., Tran, T., & Venkatesh, S. (2019). Neural Stored-program Memory. In International Conference on Learning Representations. 2020.

Although not the main contributions, the following collaborative work is the application of some work in the thesis:

- Khan, A., Le, H., Do, K., Tran, T., Ghose, A., Dam, H., & Sindhgatta, R. (2018). Memory-augmented neural networks for predictive process analytics. arXiv preprint arXiv:1802.00938.

# Abbreviations

CEC  Constant Error Carousel

CMM  Correlation Matrix Memory

CUW  Cached Uniform Writing

CVAE  Conditional Variational Autoencoder

DC-MANN  Dual Controller Memory Augmented Neural Network

DCw-MANN  Dual Controller Write-Protected Memory Augmented Neural Network

DMNC  Dual Memory Neural Computer

DNC  Differentiable Neural Computer

EMR  Electronic Medical Record

FSM  Finite-state Machine

GRU  Gated Recurrent Unit

HAM  Holographic Associative Memory

HRR  Holographic Reduced Representation

KL  Kullback–Leibler

LRUA  Least Recently Used Access

LSM  Liquid State Machine

LSTM  Long Short-Term Memory

MAED  Memory-augmented Encoder-Decoder Architecture

MANN  Memory-augmented Neural Network

MemN2N  End-to-End Memory Network

MemNN  Memory Network

MLP  Multilayer Perceptron

MoG  Mixture of Gaussians

NSM  Neural Stored-program Memory

NTM  Neural Turing Machine

NUTM  Neural Universal Turing Machine

RNN  Recurrent Neural Network

SDM  Sparse Distributed Memory

TPR  Tensor Product Representation

UTM  Universal Turing Machine

UW  Uniform Writing

VAE  Variational Autoencoder

VMED  Variational Memory Encoder-Decoder

VNA  Von Neumann Architecture

# Chapter 1

# Introduction

## 1.1 Motivations

In a broad sense, memory is the ability to store, retain and then retrieve information on request. In human brain, memory is involved in not just remembering and forgetting but also reasoning, attention, insight, abstract thinking, appreciation and imagination. Modern machine learning models find and transfer patterns from training data into some form of memory that will be utilised during inference. In the case of neural networks, long-term memories on output-input associations are stored in the weights on the connections between processing units. These connections are a simple analogy of synapses between neurons and this form of memory simulates the brain's neocortex responsible for gradual acquisition of data patterns. Learning in such scenario is slow since the signal from the output indicating how to adjust the connecting weights will be both noisy and weak (Kumaran et al., 2016). While receiving training data samples, the learning algorithm performs small update per sample to reach a global optimisation for the whole set of data.

It is crucial to keep in mind that memory in neural networks does not limit to the concept of storing associations in the observed data. For example, in sequential processes, where the individual data points are no longer independent and identically distributed (i.i.d.), some form of short-term memory must be constructed across sequence before the output is given to the network for weight updating. Otherwise, the long-term memory on associations between the output and inputs, which are given at different timestamps, will never be achieved. Interestingly, both forms of

memory are found in Recurrent Neural Networks (RNNs) (Elman, 1990; Jordan, 1997; Rumelhart et al., 1988)– a special type of neural network capable of modeling sequences. The featured short-term memory, also referred to as working memory, has been known to relate with locally stable points (Hopfield, 1982; Sussillo, 2014) or transient dynamics (Maass et al., 2002; Jaeger and Haas, 2004) of RNNs. Although these findings shed light into the formation of the working memory, the beneath memory mechanisms and how they affect the learning process remain unclear. With the rise of deep learning, more layers with complicated interconnections between neurons have been added to neural networks. These complications make it harder to understand and exploit the working memory mechanisms. Worse still, due to its short-term capacity, the working memory in RNNs struggles to cope with long sequences. These challenges require new interpretations and designs of memory for deep learning in general and RNNs in particular.

In recent years, memory-augmented neural networks (MANNs) emerge as a new form of memory construction for RNNs. They model external memories explicitly and thus, overcome the short-term limitation of the working memory. Known as one of the first attempts at representing explicit memory for RNNs, the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) stores the "world states" in a cell memory vector, which is determined after a single exposure of input at each timestep. By referring to the cell memory, LSTM can bridge longer time lags between relevant input and output events, extending the range of RNN's working memory. Recent advances have proposed new external memory modules with multiple memory vectors (slots) supporting attentional retrieval and fast-update (Graves et al., 2014; Graves et al., 2016; Weston et al., 2014). The memory slots are accessed and computed fast by a separated controller whose parameters are slowly learnt weights. Because these memories are external and separated, it is convenient to derive theoretical explanations on memorisation capacity (Gulcehre et al., 2017; Le et al., 2019). Nonetheless, with bigger memory and flexible read/write operators, these models significantly outperform other recurrent counterparts in various long-term sequential testbeds such as algorithmic tasks (Graves et al., 2014; Graves et al., 2016), reasoning over graphs (Graves et al., 2016), continual learning (Lopez-Paz et al., 2017), few-shot learning (Santoro et al., 2016; Le et al., 2020), healthcare (Le et al., 2018c; Prakash et al., 2017; Le et al., 2018b), process analytics (Khan et al., 2018), natural language understanding (Le et al., 2018a, 2019) and video question-answering (Gao et al., 2018).

In this thesis, we focus on external memory of MANNs by explaining and promoting

its influence on deep neural architectures. In the original formulation of MANNs, one controller is allowed to operate on one external memory. This simple architecture is suitable for supervised sequence labeling tasks where a sequence of inputs with target labels are provided for supervised training. However, single controller/memory design is limited for tasks involving sequence-to-sequence and especially, multi-view sequential mappings. For example, an electronic medical record (EMR) contains information on patient's admissions, each of which consists of various views such as diagnosis, medical procedure, and medicine. The complexity of view interactions, together with the unalignment and long-term dependencies amongst views poses a great challenge for classical MANNs. One important aspect of external memory is its role in imagination or generative models. Sequence generation can be supported by RNNs (Graves, 2013; Chung et al., 2015), yet how different kinds of memory in RNNs or MANNs cooperate in this process has not been adequately addressed. Another underexplored problem is to measure memorisation capacity of MANNs. There is no theoretical analysis or clear understanding on optimal operations that a memory should have to maximise its capacity. Finally, the current form of external memory is definitely not the ultimate memory mechanism for deep learning. Current MANNs are equivalent to neural simulations of Turing Machines (Graves et al., 2014). Hence, in terms of computational capacity, MANNs are not superior to RNNs, which are known to be Turing-complete (Siegelmann and Sontag, 1995). This urges new designs of external memory for MANNs that express higher computational power and more importantly, reach the capacity of human memory.

## 1.2 Aims and Scope

This thesis focuses on expanding the capacity of MANNs. Our objectives are:

- To construct a taxonomy for memory in RNNs.

- To design novel MANN architectures for modeling different aspects of memory in solving complicated tasks, which include multiple processes, generative memory, optimal operation, and universality.

- To apply such architectures to a wide range of sequential problems, especially those require memory to remember long-term contexts.

We study several practical problems that require memory:

- *Sequence to sequence mapping and multi-view sequential learning.* The former can be found in treatment recommendation where given time–ordered medical history as input, we predict a sequence of future clinical procedures and medications. The problem is harder than normal supervised sequence labeling tasks because there are dual processes: input encoding and output decoding. The latter is even more complicated as the input-output relations not only extend throughout the sequence length, but also span across views to form long-term intra-view and inter-view interactions, which is common in drug prescription and disease progression in healthcare. We aim to extend MANNs to handle these complexities, introducing generic frameworks to solve multi-view sequence to sequence mapping problems.

- *Learning generative models for sequential discrete data.* Tasks such as translation, question-answering and dialog generation would benefit from stochastic models that can produce a variety of outputs for an input. Unfortunately, current approaches using neural encoder-decoder models and their extensions using conditional variational autoencoder often compose short and dull sentences. As memory plays an important role in human imagination, we aim to use memory as a main component that blends uncertainty and variance into neural encoder-decoder models, thereby introducing variability while maintaining coherence in conversation generation.

- *Ultra-long sequential learning given limited memory resources.* Current RAM-like memory models maintain memory accessing every timesteps, thus they do not effectively leverage the short-term memory held in the controller. Previous attempts try to learn ultra-long sequences by expanding the memory, which is not always feasible and do not aim to optimise the memory by some theoretical criterion. It is critical to derive a theoretical bound on the amount of stored information and formulate an optimisation problem that maximises the bound under limited memory size constraint. Our theoretical analysis on this problem results in novel writing mechanisms that exploit the short-term memory and approximate the optimal solution.

- *Universal sequential learning.* We focus on long-life learning scenarios where sequences of tasks (subtasks) are handled by an agent, which requires a memory for tasks to avoid catastrophic forgetting. Similar situations occur when a Universal Turing Machine simulates any other Turing Machines to perform universal tasks. Inspired by the stored-program principle in computer architectures, we aim to build a Neural Stored-program Memory that enables

MANNs to switch tasks through time, adapt to variable contexts and thus fully resemble the Universal Turing Machine or Von Neumann Architecture.

## 1.3 Significance and Contribution

The significance of this thesis is organised around three central lines of work: (i) presenting taxonomy of memory in RNNs that arise under distinct roles and relations to human memory (ii) introducing novel MANN designs to model different aspects of memory and (iii) applying these designs to a wide range of practical problems in healthcare, dialog, natural language processing, few-shot, continual learning, etc. In particular, our contributions are:

- A survey for various types of memory studied for RNNs. The survey involves different forms of memory in the brain, popular memory constructions in neural networks and a taxonomy of external memory based on operational mechanisms as well as relations to computational models. Several examples of implementations by modern neural networks are also studied.

- A generic deep learning model using external memory dubbed Dual Controller Write-Protected Memory Augmented Neural Network for sequence to sequence mapping. In the encoding phase, the memory is updated as new input is read; at the end of this phase, the memory holds the history of the inputs. During the decoding phase, the memory is write–protected and the decoding controller generates one output at a time. The proposed model is demonstrated on the MIMIC-III dataset on two healthcare tasks: procedure prediction and medication prescription.

- A novel MANN architecture named Dual Memory Neural Computer (DMNC) that can model both synchronous and asynchronous dual view processes. In the modeling facet, DMNC's contributions are three-fold: (i) introducing a memory-augmented architecture for modeling multi-view sequential processes, (ii) capturing long-term dependencies and different types of interactions amongst views including intra-view, late and early inter-view interactions, and (iii) modeling multiple clinical admissions by employing a persistent memory. In the application facet, we contribute to the healthcare analytic practice by demonstrating the efficacy of DMNC on drug prescription and disease progression.

- A Variational Memory Encoder-Decoder (VMED) framework for sequence generation. VMED introduces variability into encoder-decoder architecture via the use of external memory as mixture model. By modeling the latent temporal dependencies across timesteps, our model produces a Mixture of Gaussians representing the latent distribution. We form a theoretical basis for our model formulation using mixture prior for every step of generation and apply our proposed model to conversation generation problem. The results demonstrate that VMED outperforms recent advances both quantitatively and qualitatively.

- A theory driven approach for optimising memory operations in slot-based MANNs. We contribute a meaningful measurement on MANN memory capacity. Moreover, we propose Uniform Writing (UW) and Cached Uniform Writing (CUW) as faster and optimal writing mechanisms for longer-term memorisation in MANNs. Our models are grounded in theoretical analysis on the optimality of the introduced measurement. With a comprehensive suite of synthetic and practical experiments, we provide strong evidences that our simple writing mechanisms are crucial to MANNs to reduce computation complexity and achieve competitive performance in sequence modeling tasks.

- A new type of external memory for neural networks that paves the way for a new class of MANNs that simulate Universal Turing Machines. The memory, which takes inspirations from the stored-program memory in computer architecture, gives memory-augmented neural networks a flexibility to change their control programs through time while maintaining differentiability. The mechanism simulates modern computer behavior, where CPU continually reads different instructions from RAM to execute different functions, potentially making MANNs truly neural computers.

## 1.4 Thesis Structure

This thesis contains 8 chapters with supplementary materials in the Appendix. The rest of the thesis is arranged in the following order:

- Chapter 2 presents our survey on taxonomy of memory in RNNs. The chapter first reviews various memory definitions from cognitive science. A brief introduction on the most basic neural network–Feedforward Neural Networks and their fundamental form of memory are then presented. We process to the main

part that covers Recurrent Neural Networks (RNNs) and memory categories for RNNs based on their formations. Further interpretations on memory taxonomy based on operational mechanisms and automata simulations are also investigated.

- Chapter 3 reviews a special branch of memory in RNNs and also the main focus of this thesis: memory-augmented neural networks (MANNs). We first describe the Long Short-term Memory (LSTM) and its variants. Next, we also spend a section for attention mechanism–a featured operation commonly exploited in accessing external memory in MANNs. We then introduce several advanced developments that empower RNNs with multiple memory slots, especially generic slot-based memory architectures such as Neural Turing Machine and Differentiable Neural Computer.

- Chapter 4 introduces Dual Control Memory-augmented Neural Network (DC-MANN), an extension of MANN to model sequence to sequence mapping. Our model supports write-protected decoding (DCw-MANN), which is empirically proved suitable for sequence-to-sequence task. We further extend our DC-MANN to a broader range of problems where the input can come from multiple channels. To be specific, we propose a general structure Dual Memory Neural Computer (DMNC) that can capture the correlations between two views by exploiting two external memory units. We conduct the experiments to validate the performance of these models on applications in healthcare.

- Chapter 5 presents a novel memory-augmented generation framework called Variational Memory Encoder-Decoder. Our external memory plays a role as a mixture model distribution generating the latent variables to produce the output and take part in updating the memory for future generation steps. We adapt Stochastic Gradient Variational Bayes framework to train our model by minimising variational approximation of KL divergence to accommodate the Mixture of Gaussians in the latent space. We derive theoretical analysis to backup our training protocol and evaluate our model on two open-domain and two closed-domain conversational datasets.

- Chapter 6 suggests a meaningful measurement on MANN's memory capacity. We then formulate an optimisation problem that maximises the bound on the proposed measurement. The proposed solution dubbed Uniform Writing is optimal under the assumption of equal timestep contributions. To relax this assumption, we introduce modifications to the original solution, resulting in a new solution termed Cached Uniform Writing. This method aims to balance

between memorising and forgetting via allowing overwriting mechanism. To validate the effectiveness of our solutions, we conduct experiments on six ultra-long sequential learning problems given a limited number of memory slots.

- Chapter 7 interprets MANNs as neural realisations of Turing Machines. The chapter points out a missing component–the stored-program memory, that is potential for making current MANNs truly neural computers. Then, a design of Neural Stored-program Memory (NSM) is proposed to implement stored-program principle, together with new MANN architectures that materialise Universal Turing Machines. The significance of NSM lies in its formulation as a new form of memory, standing in between slow-weight and fast-weight concepts. NSM not only induces Universal Turing Machine realisations, which imply universal artificial intelligence, but also defines another type of adaptive weights, from which other neural networks can also reap benefits.

- Chapter 8 summarises the main content of the thesis and outlines future directions.

# Chapter 2

# Taxonomy for Memory in RNNs

## 2.1 Memory in Brain

Memory is a crucial part of any cognitive model studying the human mind. This section briefly reviews memory types studied throughout the cognitive and neuroscience literature. Fig. 2.1 shows a taxonomy of cognitive memory (Kotseruba and Tsotsos, 2018).

### 2.1.1 Short-term Memory

**Sensory memory**   Sensory memory caches impressions of sensory information after the original stimuli have ended. It can also preprocess the information before transmitting it to other cognitive processes. For example, echoic memory keeps acoustic stimulus long enough for perceptual binding and feature extraction processes. Sensory memory is known to associate with temporal lope in the brain. In the neural network literature, sensory memory can be designed as neural networks without synaptic learning (Johnson et al., 2013).

**Working memory**   Working memory holds temporary storage of information related to the current task such as language comprehension, learning, and reasoning (Baddeley, 1992). Just like computer that uses RAM for its computations, the brain needs working memory as a mechanism to store and update information to perform

cognitive tasks such as attention, reasoning and learning. Human neuroimaging studies show that when people perform tasks requiring them to hold short-term memory, such as the location of a flash of light, the prefrontal cortex becomes active (Curtis and D'Esposito, 2003). As we shall see later, recurrent neural networks must construct some form of working memory to help the networks learn the task at hand. As working memory is short-term (Goldman-Rakic, 1995), the working memory in RNNs also tends to vanish quickly and needs the support from other memory mechanisms to learn complex tasks that require long-term dependencies.

### 2.1.2 Long-term Memory

**Motor/procedural memory**  The procedural memory, which is known to link to basal ganglia in the brain, contains knowledge about how to get things done in motor task domain. The knowledge may involve co-coordinating sequences of motor activity, as would be needed when dancing, playing sports or musical instruments. This procedural knowledge can be implemented by a set of if-then rules learnt for a particular domain or a neural network representing perceptual-motor associations (Salgado et al., 2012).

**Semantic memory**  Semantic memory contains knowledge about facts, concepts, and ideas. It allows us to identify objects and relationships between them. Semantic memory is a highly structured system of information learnt gradually from the world. The brain's neocortex is responsible for semantic memory and its processing is seen as the propagation of activation amongst neurons via weighted connections that slowly change (Kumaran et al., 2016).

**Episodic memory**  Episodic memory stores specific instances of past experience. Different from semantic memory, which does not require temporal and spatial information, episodic remembering restores past experiences indexed by event time or context (Tulving et al., 1972). Episodic memory is widely acknowledged to depend on the hippocampus, acting like an autoassociate memory that binds diverse inputs from different brain areas that represent the constituents of an event (Kumaran et al., 2016). It is conjectured that the experiences stored in hippocampus transfer to neocortex to form semantic knowledge as we sleep via consolidation process. Recently, many attempts have been made to integrate episodic memory into deep
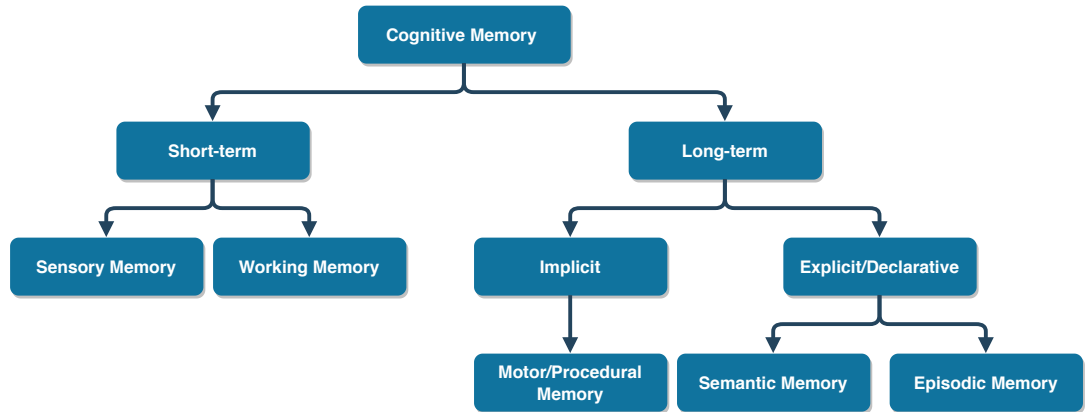
Figure 2.1: Types of memory in cognitive models

learning models and achieved promising results in reinforcement (Mnih et al., 2015; Blundell et al., 2016; Pritzel et al., 2017) and supervised learning (Graves et al., 2016; Lopez-Paz et al., 2017; Le et al., 2018b).

## 2.2 Neural Networks and Memory

### 2.2.1 Introduction to Neural Networks

**Feed-forward neural networks**

A feed-forward neural network arranges neurons in layers with connections going forward from one layer to another, creating a directed acyclic graph. That is, connections going backwards or between nodes within a layer are prohibited. Each neuron in the network is a computation unit, which takes inputs from outputs of other neurons, then applies a weighted sum followed by a nonlinear transform, and produces an output. The multilayer perceptron (MLP) is a commonly used feed-forward neural network for classifying data or approximating an unknown function. An example MLP is shown in Fig. 2.2, with three layers: input, output and a single "hidden" layer. In order to distinguish linearly inseparable data points, the activation function must be nonlinear. The weight of a connection, which resembles synapse of the neocortex, is simply a coefficient by which the output of a neuron is multiplied before being taken as the input to another neuron. Hence, the total input to a neuron $j$ is
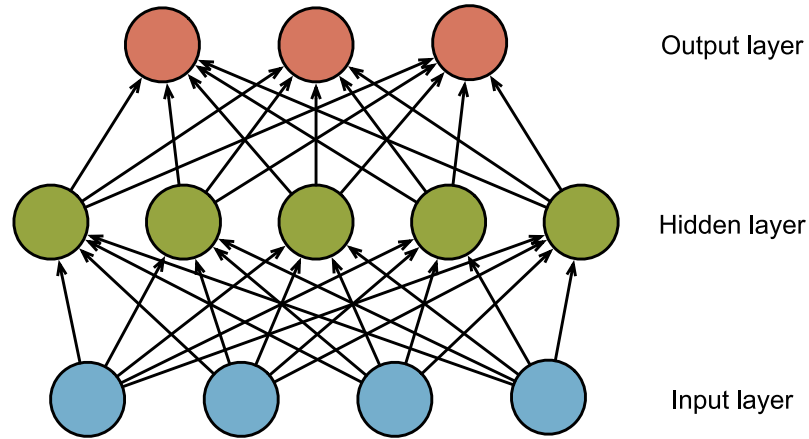
Figure 2.2: A multilayer perceptron with a single hidden-layer.

$$y_j = \sum_i w_{ij} x_i + b_j \tag{2.1}$$

where $x_i$ is the output of a neuron $i$, $w_{ij}$ is the weight of the connection from neuron $i$ to neuron $j$, and $b_j$ is a constant offset or bias. The output of neuron $j$, or $x_j$, is the result of applying an activation function to $y_j$. The following lists common activation functions used in modern neural networks,

$$\text{sigmoid}\,(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

$$\tanh\,(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.3}$$

$$\text{relu}\,(z) = \max(z, 0) \tag{2.4}$$

Given a set of training data with ground truth label for each data points, the network is typically trained with gradient-based optimisation algorithms, which estimate the parameters by minimising a loss function. A popular loss function is the average negative log likelihood

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \log P\,(\hat{y}_i = y_i | x_i) \tag{2.5}$$

where $N$ is the number of training samples, $x_i$ and $y_i$ is the $i$-th data sample

and its label, respectively, and $\hat{y}_i$ is the predicted label. During training, forward propagation outputs $\hat{y}_i$ and calculates the loss function. An algorithm called back-propagation, which was first introduced in (Rumelhart et al., 1988), computes the gradients of the loss function $\mathcal{L}$ with respect to (w.r.t) the parameters $\theta = \{w_{ij}, b_j\}$. Then, an optimisation algorithm such as stochastic gradient descent updates the parameters based on their gradients $\left\{\frac{\partial \mathcal{L}}{\partial w_{ij}}, \frac{\partial \mathcal{L}}{\partial b_j}\right\}$ as follows,

$$w_{ij} := w_{ij} - \lambda \frac{\partial \mathcal{L}}{\partial w_{ij}} \tag{2.6}$$

$$b_j := b_j - \lambda \frac{\partial \mathcal{L}}{\partial b_j} \tag{2.7}$$

where $\lambda$ is a small learning rate.

**Recurrent neural networks**

A recurrent neural network (RNN) is an artificial neural network where connections between nodes form a directed graph with self-looped feedback. This allows the network to capture the hidden states calculated so far when activation functions of neurons in the hidden layer are fed back to the input layer at every time step in conjunction with other input features. The ability to maintain the state of the system makes RNN especially useful for processing sequential data such as sound, natural language or time series signals. So far, many varieties of RNN have been proposed such as Hopfield Network (Hopfield, 1982), Echo State Network (Jaeger and Haas, 2004) and Jordan Network (Jordan, 1997). Here, for the ease of analysis, we only discuss Elman's RNN model (Elman, 1990) with single hidden layer as shown in Fig. 2.3.

An Elman RNN consists of three layers, which are input ($x \in \mathbb{R}^N$), hidden ($h \in \mathbb{R}^D$) and output ($o \in \mathbb{R}^M$) layer. At each timestep, the feedback connection forwards the previous hidden state $h_{t-1}$ to the current hidden unit, together with the values from input layer $x_t$, to compute the current state $h_t$ and output value $o_t$. The forward pass begins with a specification of the initial state $h_0$, then we apply the following
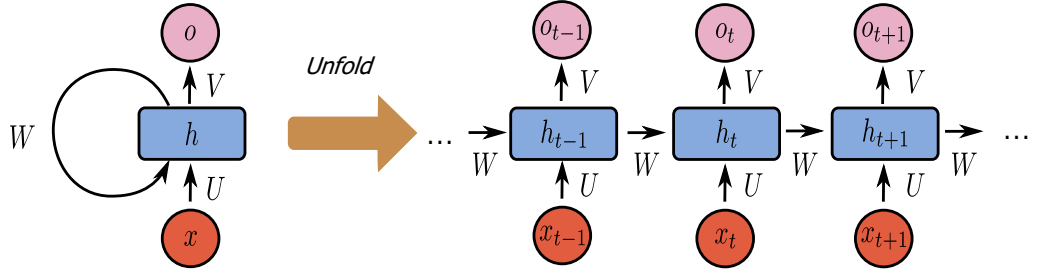
Figure 2.3: A typical Recurrent Neural Network (Left) and its unfolded representation (Right). Each neuron at timestep $t$ takes into consideration the current input $x_t$ and previous hidden state $h_{t-1}$ to generate the $t$-th output $o_t$. $W$, $U$ and $V$ are learnable weight matrices of the model.

update equations

$$h_t = f\left(h_{t-1}W + x_t U + b\right) \tag{2.8}$$

$$o_t = g\left(h_t V + c\right) \tag{2.9}$$

where $b \in \mathbb{R}^{\mathrm{D}}$ and $c \in \mathbb{R}^{\mathrm{M}}$ are the bias parameters. $U \in \mathbb{R}^{\mathrm{N}\times\mathrm{D}}$, $V \in \mathbb{R}^{\mathrm{D}\times\mathrm{M}}$ and $W \in \mathbb{R}^{\mathrm{D}\times\mathrm{D}}$ are weight matrices for input-to-hidden, hidden-to-output and hidden-to-hidden connections, respectively. $f$ and $g$ are functions that help to add non-linearity to the transformation between layers. For classification problems, $g$ is often chosen as the softmax function and the output $o_t$ represents the conditional distribution of $t$-th output given previous inputs. The final output $\hat{y}_t$ is the label whose probability score is the highest. By repeating the updates, one can map the input sequence $x = \{x_1, x_2, ..., x_T\}$ to an output sequence $\hat{y} = \{\hat{y}_1, \hat{y}_2, ..., \hat{y}_T\}$. The total loss for a given sequence $x$ paired with a ground-truth sequence $y = \{y_1, y_2, ..., y_T\}$ would then be the sum of the losses over all the timesteps

$$\mathcal{L}\left(y|x\right) = \sum_{t=1}^{T}\mathcal{L}_t\left(y_t|x_1, x_2, ..., x_t\right) = -\sum_{t=1}^{T}\log P\left(\hat{y}_t = y_t|x_1, x_2, ..., x_t\right)$$

The loss function can be minimised by using gradient descent approach. The derivatives w.r.t the parameters can be determined by the Back-Propagation Through Time algorithm (Werbos, 1990). RNNs are widely used in sequential tasks such as language modeling (Mikolov et al., 2010), handwriting generation (Graves, 2013) and speech recognition (Graves et al., 2013). RNNs demonstrate better performance than other classical approaches using Hidden Markov Model (HMM) or Conditional Random Fields (CRFs).

## 2.2.2   Semantic Memory in Neural Networks

Neural networks learn structured knowledge representation from the data by adjusting connection weights amongst the units in the network under supervised training paradigms (Hinton et al., 1986; Rumelhart et al., 1988; Plunkett and Sinha, 1992). The connection weights capture the semantic structure of the domain under modeling (McClelland et al., 1995; Rogers and McClelland, 2004). The trained model generalises to novel examples rather than just naively memorising training items. However, modern deep learning models are often massively over-parameterised and thus prone to overfitting, even to noise (Zhang et al., 2016b). Further investigations indicate that although deep networks may employ brute-force memorising strategy, they should operate in a fashion that can perform inductive generalisation (Arpit et al., 2017; Krueger et al., 2017). Unfortunately, since all of these arguments are validated empirically or via simulations, no theoretical principles governing semantic knowledge extraction were given.

The lack of theoretical guarantee remained until recently when Saxe et al. (2019) confirmed the existence of semantic memory in neural network by theoretically describing the trajectory of knowledge acquisition and organisation of neural semantic representations. The paper is restricted to a simple linear neural network with one hidden layer. The network is trained to correctly output the associated properties or features of the input items (e.g., dog →bark, horse →big). Each time a training sample $i$ is presented as $\{x_i, y_i\}$, the weights of the network $W_1$ and $W_2$ are adjusted by a small amount to gradually minimise the squared error loss $\mathcal{L} = \|y_i - \hat{y}_i\|^2$. The parameter update rule is derived via standard back propagation as follows,

$$\Delta W_1 = \lambda W_2^\top \left( y_i - \hat{y}_i \right) x_i^\top \tag{2.10}$$

$$\Delta W_2 = \lambda \left( y_i - \hat{y}_i \right) \left( W_1 x_i \right)^\top \tag{2.11}$$

where $\lambda$ is the learning rate. We are interested in estimating the total weight change after epoch $t$, which can be approximated, when $\lambda \ll 1$, as the following,

$$\Delta W_1 \left( t \right) \approx \lambda P W_2 \left( t \right)^\top \left( \Sigma^{yx} - W_2 \left( t \right) W_1 \left( t \right) \Sigma^x \right) \tag{2.12}$$

$$\Delta W_2 \left( t \right) \approx \lambda P \left( \Sigma^{yx} - W_2 \left( t \right) W_1 \left( t \right) \Sigma^x \right) W_1 \left( t \right)^\top \tag{2.13}$$

where $P$ is the number of training samples; $\Sigma^x = E\left[xx^\top\right]$ and $\Sigma^{yx} = E\left[yx^\top\right]$ are input and input-output correlation matrices, respectively. We can take the continuum limit of this difference equation to obtain the following system of differential equations

$$\tau \frac{d}{dt} W_1 = W_2^\top \left(\Sigma^{yx} - W_2 W_1 \Sigma^x\right) \tag{2.14}$$

$$\tau \frac{d}{dt} W_2 = \left(\Sigma^{yx} - W_2 W_1 \Sigma^x\right) W_1^\top \tag{2.15}$$

where $\tau = \frac{1}{P\lambda}$. To simplify the equations, we assume $\Sigma^x = I$ and apply reparametrisation trick to obtain

$$\tau \frac{d}{dt} \overline{W}_1 = \overline{W}_2^\top \left(S - \overline{W}_2 \overline{W}_1\right) \tag{2.16}$$

$$\tau \frac{d}{dt} \overline{W}_2 = \left(S - \overline{W}_2 \overline{W}_1\right) \overline{W}_1^\top \tag{2.17}$$

where $S$ is the diagonal matrix in the singular value decomposition of $\Sigma^{yx} = USV^\top$; $\overline{W}_1$ and $\overline{W}_2$ are new variables such that $W_1 = R\overline{W}_1 V^\top$ and $W_2 = U\overline{W}_2 R$ with an arbitrary orthogonal matrix $R$. When $\overline{W}_1(0)$ and $\overline{W}_2(0)$ are initialised with small random weights, we can approximate them with diagonal matrices of equal modes. A closed form solution of the scalar dynamic corresponding to each mode of Eqs. (2.16) and (2.17) can be derived as follows,

$$a_\alpha(t) = \frac{s_\alpha e^{2s_\alpha t/\tau}}{e^{2s_\alpha t/\tau} - 1 + s_\alpha/a_\alpha(0)} \tag{2.18}$$

where $a_\alpha$ is a diagonal element of the time-dependent diagonal matrix $A(t)$ such that $A(t) = \overline{W}_2(t) \overline{W}_1(t)$ . Inverting the change of variables yields

$$W_1(t) = Q\sqrt{A(t)}V^\top \tag{2.19}$$

$$W_2(t) = U\sqrt{A(t)}Q^{-1} \tag{2.20}$$

where $Q$ is an arbitrary invertible matrix. If the initial weights are small, then the matrix $Q$ will be close to a rotation matrix. Factoring out the rotation, the hidden

representation of item $i$ is

$$h_i^\alpha(t) = \sqrt{a_\alpha(t)} v_i^\alpha \tag{2.21}$$

where $v_i^\alpha = V^\top [\alpha, i]$. Hence, we obtain a temporal evolution of internal representations $h$ of the deep network. By using multi-dimensional scaling (MDS) visualisation of the evolution of internal representations over developmental time, Saxe et al. (2019) demonstrated a progressive differentiation of hierarchy in the evolution, which matched the data's underlying hierarchical structure. When we have the explicit form of the evolution (Eq. (2.21)), this matching can be proved as an inevitable consequence of deep learning dynamics when exposed to hierarchically structured data (Saxe et al., 2019).

### 2.2.3   Associative Neural Networks

Associative memory is used to store associations between items. It is a general concept of memory that spans across episodic, semantic and motor memory in the brain. We can use neural networks (either feed-forward or recurrent) to implement associative memory. There are three kinds of associative networks:

- Heteroassociative networks store $Q$ pair of vectors $\{x^1 \in \mathcal{X}, y^1 \in \mathcal{Y}\}, ..., \{x^Q \in \mathcal{X}, y^Q \in \mathcal{Y}\}$ such that given some key $x^k$, they return value $y^k$.

- Autoassociative networks are a special type of the heteroassociative networks, in which $y^k = x^k$ (each item is associated with itself).

- Pattern recognition networks are also a special case where $x^k$ is associated with a scalar $k$ representing the item's category.

Basically, these networks are used to represent associations between two vectors. After two vectors are associated, one can be used as a cue to retrieve the other. In principle, there are three functions governing an associative memory:

- Encoding function $\otimes : \mathcal{X} \times \mathcal{Y} \to \mathcal{M}$ associates input items into some form of memory trace $\mathcal{M}$.

- Trace composition function $\oplus : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ combines memory traces to form the final representation for the whole dataset.

- Decoding function $\bullet : \mathcal{X} \times \mathcal{M} \to \mathcal{Y}$ produces a (noisy) version of the item given its associated.

Different models employ different kinds of functions (linear, non-linear, dot product, outer product, tensor product, convolution, etc.). Associative memory concept is potential to model memory in the brain (Marr and Thach, 1991). We will come across some embodiment of associative memory in the form of neural networks in the next sections.

## 2.3 The Constructions of Memory in RNNs

### 2.3.1 Attractor dynamics

Attractor dynamics denotes neuronal network dynamics which is dominated by groups of persistently active neurons. In general, such a persistent activation associates with an attractor state of the dynamics, which for simplicity, can take the form of fixed-point (Amit, 1992). This kind of network can be used to implement associative memory by allowing the network's attractors to be exactly those vectors we would like to store (Rojas, 2013). The approach supports memory for the items per se, and thus differs from semantic memory in the sense that the items are often stored quickly and what being stored cannot represent the semantic structure of the data. Rather, attractor dynamics resembles working and episodic memory. Like episodic memory, it acts as an associative memory, returning stored value when triggered with the right clues. The capacity of attractor dynamics is low, which reflects the short-term property of working memory. In the next part of the sub-section, we will study these characteristics through one embodiment of attractor dynamics.

**Hopfield network**

The Hopfield network, originally proposed in 1982 (Hopfield, 1982), is a recurrent neural network that implements associative memory using fix-points as attractors. The function of the associative memory is to recognise previously learnt input vectors, even in the case where some noise has been added. To achieve this function, every neuron in the network is connected to all of the others (see Fig. 2.4 (a)).

Each neuron outputs discrete values, normally 1 or $-1$, according to the following equation

$$x_i\left(t+1\right) = \text{sign}\left(\sum_{j=1}^{N} w_{ij}x_j\left(t\right)\right) \tag{2.22}$$

where $x_i\left(t\right)$ is the state of $i$-th neuron at time $t$ and $N$ is the number of neurons. Hopfield network has a scalar value associated with the state of all neurons $x$, referred to as the "energy" or Lyapunov function,

$$E\left(x\right) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}x_i x_j \tag{2.23}$$

If we want to store $Q$ patterns $x^p$, $p = 1, 2, ..., Q$, we can use the Hebbian learning rule (Hebb, 1962) to assign the values of the weights as follows,

$$w_{ij} = \sum_{p=1}^{Q} x_i^p x_j^p \tag{2.24}$$

which is equivalent to setting the weights to the elements of the correlation matrix of the patterns[1].

Upon presentation of an input to the network, the activity of the neurons can be updated (asynchronously) according to Eq. (2.22) until the energy function has been minimised (Hopfield, 1982). Hence, repeated updates would eventually lead to convergence to one of the stored patterns. However, the network will possibly converge to spurious patterns (different from the stored patterns) as the energy in these spurious patterns is also a local minimum.

**The capacity problem**

The memorisation of some pattern can be retrieved when the network produces the desired vector $x^p$ such that $x\left(t+1\right) = x\left(t\right) = x^p$. This happens when the crosstalk computed by

---

[1]As an associative memory, Hopfield network implements $\otimes$, $\oplus$, $\bullet$ by outer product, addition and nonlinear recurrent function, respectively.

(a) Hopfield Network  (b) Liquid State Machine
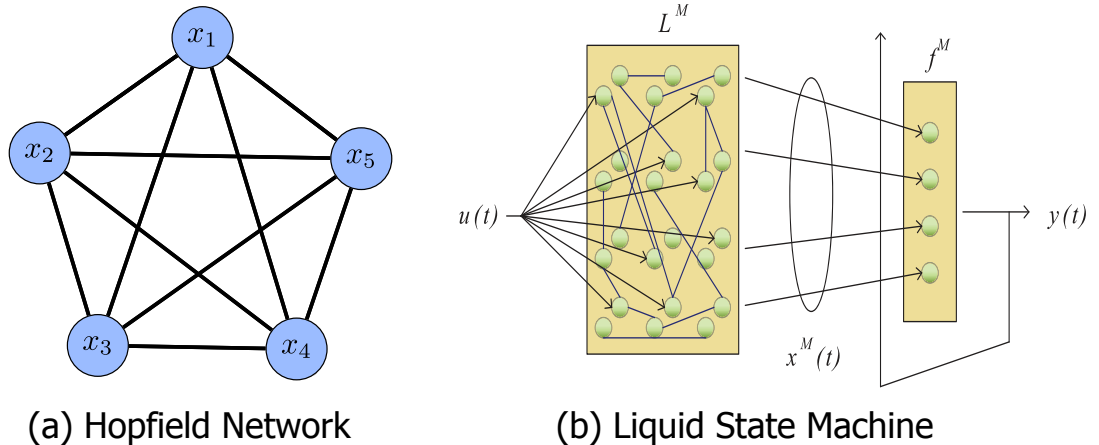
Figure 2.4: (a) Hopfield network with five neurons. (b) Structure of a Liquid State Machine $M$. The machine wants to transform input stream $u(\cdot)$ into output stream $y(\cdot)$ using some dynamical system $L^M$ (the liquid).

$$\sum_{q=1,q\neq p}^{Q} x^q \left(x^p \cdot x^q\right) \tag{2.25}$$

is less than $N$. If the crosstalk term becomes too large, it is likely that previously stored patterns are lost because when they are presented to the network, one or more of their bits are flipped by the associative computation. We would like to keep the probability that this could happen low, so that stored patterns can always be recalled. If we set the upper bound for one bit failure at 0.01, the maximum capacity of the network is $Q \approx 0.18N$ (Rojas, 2013). With this low capacity, RNNs designed as attractor dynamics have difficulty handling big problems with massive amount of data.

## 2.3.2 Transient Dynamics

One major limitation of memorising by attractor mechanisms is the incapability of remembering sequences of past inputs. This demands a new paradigm to explain the working memory mechanism that enable RNNs to capture sequential dependencies and memorise information between distance external stimuli. Within this new paradigm, the trajectories of network states should become the main carriers of information about external sensory stimuli. Recent proposals (Maass et al., 2002; Maass, 2011; Jaeger and Haas, 2004) have suggested that an arbitrary recurrent network could store information about recent input sequences in its transient dynamics despite the presence of attractors (the pattern might or might not converge to the

attractors). A useful analogy is the surface of a liquid. Transient ripples on the surface can encode information about past objects that were thrown in even though the water surface has no attractors (Ganguli et al., 2008). In the light of transient dynamics, RNNs carry past information to serve a given task as a working memory.

**Liquid State Machines**

Liquid State Machines (LSMs) (Maass et al., 2002) use a dynamic reservoir/liquid $(L^M)$, which consists of nodes randomly connected to each other, to handle time-series data. The purpose is to map an input function of time $u(t)$–a continuous sequence of disturbances, to an output function $y(t)$ that provides a real-time analysis of the input sequence. In order to achieve that, we assume that at every time $t$, $L^M$ generates an internal "liquid state" $x^M(t)$, which constitutes its current response to preceding perturbations $u(s)$ for $s \leq t$. After a certain time-period, the state of the liquid $x^M(t)$ is read as input for a readout network $f^M$, which by assumption, has no temporal integration capability of its own. This readout network learns to map the states of the liquid to the target outputs as illustrated in Fig. 2.4 (b).

All information about the input $u(s)$ from preceding time points $s \leq t$ that is needed to produce a target output $y(t)$ at time $t$ has to be contained in the current liquid state $x^M(t)$. LSMs allow realisation of large computational power on functions of time even if all memory traces are continuously decaying. Instead of worrying about the code and location where information about past inputs is stored, the approach focuses on addressing the separation question: for which later time point $t$ will any two significantly different input functions of time $u(t)$ and $v(t)$ cause significantly different liquid states $x_u^M(t)$ and $x_v^M(t)$ (Maass, 2011).

Most implementations of LSMs use the reservoir of untrained neurons. In other words, there is no need to train the weights of the RNN. The recurrent nature of the connections fuses the input sequence into a spatio-temporal pattern of neuronal activation in the liquid and computes a large variety of nonlinear functions on the input. This mechanism is theoretically possible to perform universal continuous computations. However, separation and approximation properties must be fulfilled for the system to work well. Similar neural network design can be found in Echo state networks (Jaeger and Haas, 2004). A Liquid State Machine is a particular kind of spiking neural networks that more closely mimics biological neural networks (Maass, 1997).

**Memory trace of recurrent networks**

When viewing recurrent networks as transient dynamics, one may want to measure the lifetimes of transient memory traces in the networks. Ganguli et al. (2018) studied a discrete time network whose dynamics is given by

$$x_i(n) = f\left([Wx(n-1)]_i + v_i s(n) + z_i(n)\right), \, i = 1, ..., N \quad (2.26)$$

Here, a scalar time-varying signal $s(n)$ drives an RNN of $N$ neurons. $x(n)$ is the network state at $n$-th timestep, $f(\cdot)$ is a general sigmoidal function, $W$ is an $N \times N$ recurrent connectivity matrix, and $v$ is a vector of feed-forward connections encoding the signal into the network. $z(n)$ denotes a zero mean Gaussian white noise with covariance $\langle z_i(k_1), z_j(k_2) \rangle = \varepsilon \delta_{k_1, k_2} \delta_{i,j}$.

The authors built upon Fisher information to construct useful measures of the efficiency with which the network state $x(n)$ encodes the history of the signal $s(n)$, which can be derived as

$$J(k) = v^\top W^{k\top} \left(\varepsilon \sum_{k=0}^{\infty} W^k W^{k\top}\right)^{-1} W^k v \quad (2.27)$$

where $J(k)$ measures the Fisher information that $x(n)$ retains about a signal entering the network at $k$ time steps in the past. For a special case of normal networks having a normal connectivity matrix $W$, Eq. (2.27) simplifies to

$$J(k) = \sum_{i=1}^{N} v_i^2 |\lambda_i|^{2k} \left(1 - |\lambda_i|^2\right) \quad (2.28)$$

where $\lambda_i$ is the $i$-th eigenvalue of $W$. For large k, the decay of the Fisher information is determined by the magnitudes of the largest eigenvalues and it decays exponentially. Similar findings with different measurements on the memory trace in modern recurrent networks are also found in a more recent work (Le et al., 2019).

## 2.4   External Memory for RNNs

Recurrent networks can in principle use their feedback connections to store representations of recent input events in the form of implicit memory (either attractor

or transient dynamics). Unfortunately, from transient dynamics perspective, the implicit memory tends to decay quickly (Ganguli et al., 2008; Le et al., 2019). This phenomenon is closely related to gradient vanishing/exploding problems (Bengio et al., 1994; Hochreiter and Schmidhuber, 1997; Pascanu et al., 2013) which often occur when training RNNs with gradient-based algorithms such as Back-Propagation Through Time (Williams and Zipser, 1989; Werbos, 1990). A solution is to equip RNNs with external memory to cope with exponential decay of the implicit short-term memory. The external memory enhances RNNs with stronger working (Hochreiter and Schmidhuber, 1997; Cho et al., 2014b; Graves et al., 2014, 2016) or even episodic-like memory (Graves et al., 2014; Santoro et al., 2016). We will spend the next sections to analyse different types of external memory and their memory operation mechanisms. Examples of modern recurrent neural networks that utilise external memory are also discussed.

## 2.4.1 Cell Memory

Despite the fact that RNNs offer working memory mechanisms to handle sequential inputs, learning what to put in and how to utilise the memory is challenging. Back-Propagation Through Time (Williams and Zipser, 1989; Werbos, 1990) is the most common learning algorithm for RNNs, yet it is inefficient in training long sequences mainly due to insufficient or decaying backward error signals. This section will review the analysis of this problem and study a group of methodologies that overcome the problem through the use of cell memory and gated operation.

**Hochreiter's analysis on gradient vanishing/exploding problems**

Let us assume that the hidden layer of an RNN has $n$ neurons. With differentiable activation function $f_i$, the activation of a neuron $i$ at step $t$ of the recurrent computation is as follow,

$$y^i(t) = f_i\left(\sum_j w_{ij} y^j(t-1)\right) \tag{2.29}$$

$$= f_i(net_i(t)) \tag{2.30}$$
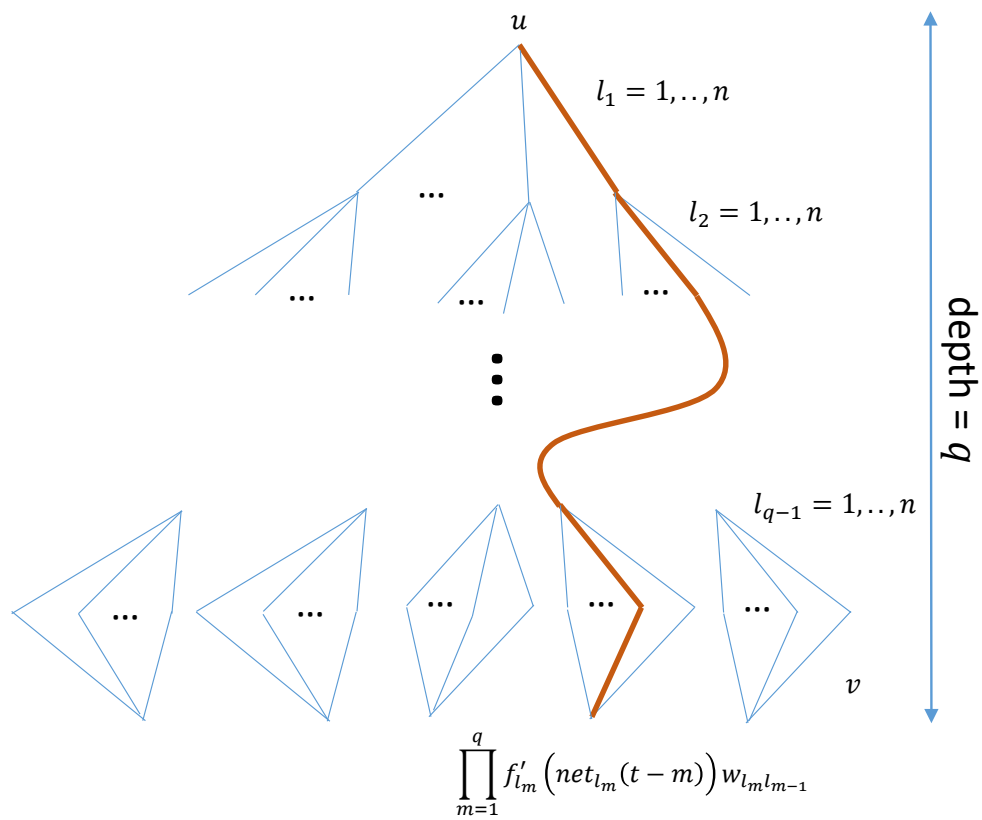
Figure 2.5: Error back flow from $\vartheta_u(t)$ to $\vartheta_v(t-q)$ in the computation graph. Each computation node has $n$ children. Each product term corresponds to a computation path of depth $q$ from node $u$ to $v$. The sum of $n^{q-1}$ products is the total error.

The backpropagated error signal for neuron $j$ at step $t$ is

$$\vartheta_j(t) = f_j'(net_j(t)) \sum_i w_{ij} \vartheta_i(t+1) \tag{2.31}$$

The error occurring at an arbitrary neuron $u$ at time step $t$ $(\vartheta_u(t))$ is backpropagated through time for $q$ timesteps to an arbitrary neuron $v$ $(\vartheta_v(t-q))$. We can measure the contribution of the former to the latter as the following,

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \begin{cases} f_v'(net_v(t-1)) w_{uv} & ; q = 1 \\ f_v'(net_v(t-q)) \sum_{l=1}^{n} \frac{\partial \vartheta_l(t-q+1)}{\partial \vartheta_u(t)} w_{lv} & ; q > 1 \end{cases} \tag{2.32}$$

By induction, we can obtain expressive form of the recursive Eq. (2.32) as

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \sum_{l_1=1}^{n} \cdots \sum_{l_{q-1}=1}^{n} \prod_{m=1}^{q} f_{l_m}'(net_{l_m}(t-m)) w_{l_m l_{m-1}} \tag{2.33}$$

where $l_q = v$ and $l_0 = u$. The computation can be visually explained through a drawing of the computation graph as in Fig. 2.5.

It is obvious to realise that if $\left| f_{l_m}'(net_{l_m}(t-m)) w_{l_m l_{m-1}} \right|$ is greater (smaller) than 1 for all $m$, then the largest product increases (decreases) exponentially with $q$, which represents the exploding and vanishing gradient problems in training neural networks. These problems are critical since they prevent proper update on the weights of the model, and thus freeze or disturb the learning process. With nonlinear activation functions such as sigmoid, the term $\left| f_{l_m}'(net_{l_m}) w_{l_m l_{m-1}} \right|$ goes to zero when $w_{l_m l_{m-1}} \to \infty$ and is less than 1 when $\left| w_{l_m l_{m-1}} \right| < 4$, which implies vanishing gradient tends to occur with nonlinear activation function.

We can also rewrite Eq. (2.32) in matrix form for $q > 1$ as follows,

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = W_u^\top F'(t-1) \prod_{m=2}^{q-1} (W F'(t-m)) W_v f_v'(net_v(t-q)) \tag{2.34}$$

where the weight matrix $W$ have its elements $W_{ij} = w_{ij}$. $W_u$ and $W_v$ are $u$'s incoming weight vector and $v$'s outgoing weight vector, respectively, such that $[W_u]_i = w_{ui}$ and $[W_v]_i = w_{vi}$. $F'(t-m)$ is a diagonal matrix whose diagonal elements $F'(t-m)_{ii} = f_i'(net_i(t-m))$. Using a matrix norm $\|\cdot\|_A$ compatible with vector norm $\|\cdot\|_p$, we

define

$$f'_{max} := \max_{m=1,\ldots,q} \left\{ \left\| F'(t-m) \right\|_A \right\} \tag{2.35}$$

By applying norm sub-multiplicativity and using the inequality

$$\left| x^T y \right| \leq n \left\| x \right\|_\infty \left\| y \right\|_\infty \leq n \left\| x \right\|_p \left\| y \right\|_p,$$

we obtain a weak upper bound for the contribution

$$\left| \frac{\partial \vartheta_v (t-q)}{\partial \vartheta_u (t)} \right| \leq n \left( f'_{max} \left\| W \right\|_A \right)^q \tag{2.36}$$

This result confirms the exploding and vanishing gradient problems since the error backprob contribution decays (when $f'_{max} \left\| W \right\|_A < 1$) or grows (when $f'_{max} \left\| W \right\|_A > 1$) exponentially with $q$. More recent analyses on the problems are presented by Bengio et al., (1994) and Pascanu et al., (2013).

**Problem with naive solution**

When analysing a single neuron $j$ with a single connection to itself, avoiding the exploding and vanishing gradient problems requires

$$f'_j \left( net_j (t) \right) w_{jj} = 1 \tag{2.37}$$

In this case, the constant error flow is enforced by using linear function $f_j$ and constant activation (e.g., $f_j (x) = x$ with $\forall x$ and setting $w_{jj} = 1$). These properties are known as the *constant error carousel* (CEC). The strict constraint makes this solution unattractive because it limits computation capacity of RNNs with linear activation. Even worse, neuron $j$ is connected to other neurons as well, which makes thing complicated. Let us consider an additional input weight $w_{ji}$ connecting neuron $i$ to $j$. $w_{ji}$ is learnt to keep relevant external input from $i$ such that $w_{ji} y_i > 0$ when the input signal $y_i$ is relevant. Assume that the loss function is reduced by keeping neuron $j$ active ($> 0$) for a while between two occurrences of two relevant inputs. During that period, activation of neuron $j$ is possibly disturbed since with a fixed $w_{ji}$, $w_{ji} y_i < 0$ with irrelevant inputs. Since $y^j (t) = f_j \left( w_{jj} y^j (t-1) + w_{ji} y^i (t-1) \right)$

where $f_j$ is linear, $y^j(t-1)$ is kept constant and $y^i(t-1)$ scales with the external input, it is likely to deactivate neuron $j$. Hence, if naively following CEC, learning a $w_{ji}$ to capture relevant inputs while protecting neuron $j$ from disturbances of irrelevant inputs is challenging (input weight conflict (Hochreiter and Schmidhuber, 1997)). Similar problem happens with the output weight (output weight conflict). These conflicts make the learning hard, and require a more flexible mechanism for controlling input/output weight impact conditioned on the input signal.

**The original Long Short-Term Memory (LSTM)**

Hochreiter and Schmidhuber (1997) originally proposed LSTM using multiplicative gate units and a memory cell unit to overcome the weight conflicts while following CEC. The idea is to apply CEC to neurons specialised for memorisation, each of which has an internal state independent from the activation function. This separation between memorisation and computation is essential for external memory concept. Besides, to control input/output weight impact, gate units conditioned on the inputs are multiplied with the incoming/outgoing connections, modifying the connection value through time. In particular, if a neuron $c_j$ becomes cell memory, its output is computed as

$$y^{c_j}(t) = y^{out_j}(t)\, h\left(s_{c_j}(t)\right) \tag{2.38}$$

where $y^{out_j}(t)$ is the output gate, $h$ is a differentiable function for scaling down the neuron's output, and $s_{c_j}$ captures past information by using the dynamics

$$s_{c_j}(0) = 0 \tag{2.39}$$

$$s_{c_j}(t) = y^{fg_j}(t)\, s_{c_j}(t-1) + y^{in_j}(t)\, f\left(net_{c_j}(t)\right) \text{ for } t > 0 \tag{2.40}$$

where $y^{in_j}(t)$ is the input gate, $y^{fg_j}(t)$ is the (optional) forget gate and $f$ is the activation function, which can be nonlinear. Without forget gate, $c_j$ can be viewed as a neuron with an additional fixed self-connection. The computation paths that mainly pass through this special neuron preserve the backward error. The remaining problem is to protect this error from disturbance from other paths. The gates are calculated as

$$y^{g_j}(t) = f_{g_j}\left(\sum_u w_{g_j u} y^u(t-1)\right) \qquad (2.41)$$

where $g$ can represent input, output and forget gate. The gates are adaptive according to the input from other neurons, hence, it is possible to learn $\{w_{g_j u}\}$ to resolve the input/output weight conflict problem.

Although the cell memory provides a potential solution to cope with training RNN over long time lag, unfortunately, in practice, the multiplicative gates are not good enough to overcome a fundamental challenge of LSTM: the gates are not coordinated at the start of training, which can cause $s_{c_j}$ to explode quickly (internal state drift). Various variants of LSTM have been proposed to tackle the problem (Greff et al., 2016). We will review some of them in Chapter 3.

### Cell memory as external memory

From Eq. (2.40), we can see the internal state of the cell memory holds two types of information: (i) the previous cell state and (ii) the normal state of RNN, which is the activation of current computation. Therefore, the cell state contains a new form of external memory for RNNs. The size of the memory is often equal the number of hidden neurons in RNNs and thus, cell memory is also known as vector memory. The memory supports writing and reading mechanisms implemented as gated operations in $y^{in_j}(t)$ and $y^{out_j}$, respectively. They control how much to write to and read from the cell state. With the cell state, which is designed to keep information across timesteps, the working memory capacity of LSTM should be greater than that of RNNs. The memory reading and writing are also important to determine the memory capacity. For instance, if writing irrelevant information too often, the content in the cell state will saturate and the memory fails to hold much information. Later works make use of the gating mechanism to build skip-connections between inputs (a source of raw memory) and neurons in higher layers (Srivastava et al., 2015; He et al., 2016), opening chance to ease the training of very deep networks.

## 2.4.2 Holographic Associative Memory

The holographic associative memory (HAM) roots its operation on the principle of optical holography, where two beams of light are associated with one another in a holograms such that reconstruction of one original beam can be made by presenting another beam. Recall that the capacity of associative memory using attractor dynamics is low. To maintain $Q$ pairs of key-value (in Hopfield network, value is also key), it requires $N^2$ weight storage where $Q \approx 0.18N$. HAM presents a solution to compress the key-values into a fixed size vector via Holographic Reduced Representation (HRR) without substantial loss of information (Plate, 1995). This can be done in real or complex domain using circular convolution or element-wise complex multiplication for the encoding function ($\otimes$), respectively. The compressed vector ($\mathcal{M}$), as we shall see, can be used as external memory for RNNs.

**Holographic Reduced Representation**

Consider a complex-valued vector key $x \in \mathbb{C}^N$,

$$x = \left[ x_a\left[1\right] e^{ix_\phi[1]}, ..., x_a\left[N\right] e^{ix_\phi[N]} \right] \tag{2.42}$$

The association encoding is computed by

$$m = x \circledast y \tag{2.43}$$
$$= \left[ x_a\left[1\right] y_a\left[1\right] e^{i\left(x_\phi[1]+y_\phi[1]\right)}, ..., x_a\left[N\right] y_a\left[N\right] e^{i\left(x_\phi[N]+y_\phi[N]\right)} \right] \tag{2.44}$$

where $\circledast$ is element-wise complex multiplication, which multiplies the moduli and adds the phases of the elements. Trace composition function is simply addition

$$m = x^1 \circledast y^1 + x^2 \circledast y^2 + ... + x^Q \circledast y^Q \tag{2.45}$$

Although the memory $m$ is a vector with the same dimension as that of stored items, it can store many pairs of items since we only need to store the information that discriminates them. The decoding function is multiplying an inverse key $x^{-1} = \left[ x_a\left[1\right]^{-1} e^{-ix_\phi[1]}, ..., x_a\left[N\right]^{-1} e^{-ix_\phi[N]} \right]$ with the memory as follows,

$$\tilde{y} = x^{-1} \circledast m \tag{2.46}$$

$$= x^{-1} \circledast \left( \sum_{\forall k} x^k \circledast y^k \right) \tag{2.47}$$

$$= y + x^{-1} \circledast \left( \sum_{\forall k : x^k \neq x} x^k \circledast y^k \right) \tag{2.48}$$

The second term in Eq. (2.48) is noise and should be minimised. Under certain conditions, the noise term has zero mean (Plate, 1995). One way to reconstruct better is to pass the retrieved vector through an auto-associative memory to correct any errors.

**Redundant Associative Long Short-Term Memory**

One recent attempt to apply HRR to LSTM is the work by Danihelka et al. (2016). The authors first propose Redundant Associative Memory, an extension of HRR with multiple memory traces for multiple transformed copies of each key vector. In particular, each key vector will be transformed $S$ times using $S$ constant random permutation matrix $P_s$. Hence, we obtain the memory trace $c_s$ for the $s$-th copy

$$c_s = \sum_{\forall k} \left( P_s x^k \right) \circledast y^k \tag{2.49}$$

The $k$-th value is retrieved as follows,

$$\tilde{y}^k = \frac{1}{S} \sum_{s=1}^{S} \left( \overline{P_s x^k} \right) \circledast c_s \tag{2.50}$$

$$= y^k + \sum_{k' \neq k} y^{k'} \circledast \frac{1}{S} \sum_{s=1}^{S} P_s \left[ \overline{x^k} \circledast x^{k'} \right] \tag{2.51}$$

where $\overline{P_s x^k}$ and $\overline{x^k}$ are the complex conjugates of $P_s x^k$ and $x^k$, respectively, which are equal to the inverses if the modulus $x_a^k = 1$. Since permuting the key decorrelates the retrieval noise, the noise term has variance $O\left(\frac{Q}{S}\right)$ and increase the number of copies will enhance retrieval quality.

Applying the idea to LSTM, we can turn the cell memory to a holographic memory

by encoding the term containing input activation in Eq. (2.40) before added up to the cell memory. The network learns to generate the key $x^k$ and the inverse key $(x^{-1})^k$ for $k$-th timestep. It should be noted that the inverse key at $k$-th timestep can associate to some preceding key. Following Redundant Associative Memory extension, multiple copies of cell memory are employed. The cell state will be decoded to retrieve some past input activation necessary for current output (Danihelka et al., 2016). Then the decoded value will be multiplied with the output gate as in Eq. (2.38).

### 2.4.3 Matrix Memory

**Correlation matrix memory**

Correlation Matrix Memory (CMM) stores associations between pairs of vectors using outer product as the encoding function. Although the purpose looks identical to that of attractor dynamics, CMM is arranged differently using feed-forward neural network without self-loop connections. The memory construction ($\otimes + \oplus$) follows Hebbian learning

$$M = \sum_{i=1}^{Q} y_i x_i^\top \tag{2.52}$$

where $Q$ is the number of stored patterns, $x_i$ and $y_i$ are the $i$-th key-value pair. The memory retrieval ($\bullet$) is simply dot product

$$\tilde{y}_j = M x_j \tag{2.53}$$

$$= \left( \sum_{i=1}^{Q} y_i x_i^\top \right) x_j \tag{2.54}$$

$$= \sum_{i=1,i\neq j}^{Q} y_i x_i^\top x_j + y_j \|x_j\|^2 \tag{2.55}$$

If the keys are orthonormal, then the retrieval is exact. Actually, linear independence is enough for exact retrieval. In this case, WidrowHoff learning rule should be used.

When the stored values are binary vectors, a threshold function is applied. The

capacity for binary CMM is heavily dependent on the sparsity of the patterns (the sparser the better). In general, CMM offers a capacity that is at least comparable to that of the Hopfield model (Baum et al., 1988).

**Fast-weight**

Fast-weights refer to synapses that change slower than neuronal activities but much faster than the standard slow weights. These fast weights form temporary memories of the recent past that support the working memory of RNNs (Hinton and Plaut, 1987; Schmidhuber, 1992; Ba et al., 2016). In a recent fast-weight proposal (Ba et al., 2016), the memory is similar to a correlation matrix memory with decaying factor to put more weight on the recent past. In particular, the fast memory weight matrix $A$ is computed as follows,

$$A\left(t\right) = \lambda A\left(t-1\right) + \eta h\left(t\right) h\left(t\right)^{\top} \tag{2.56}$$

where $\lambda$ and $\eta$ are the decay and learning rate, respectively. $h\left(t\right)$ is the hidden state of the RNN and also the pattern being stored in the associative memory. The memory is used to iteratively refine the next hidden state of RNN as the following,

$$h_{s+1}\left(t+1\right) = f\left(\left[Wh\left(t\right) + Cx\left(t\right)\right] + A\left(t\right) h_s\left(t+1\right)\right) \tag{2.57}$$

where $h_0\left(t+1\right) = f\left(Wh\left(t\right) + Cx\left(t\right)\right)$, following the ordinary dynamics of RNNs and $h_s\left(t+1\right)$ is the hidden state at $s$-th step of refinement.

**Tensor product representation**

Tensor product representation (TPR) is a mechanism to store symbolic structures. It shares common properties with CMM when the tensor is of order 2, in which tensor product is equivalent to outer product. In TPR, relations between concepts are described by the set of filler-role bindings. The vector space of filler and role are denoted as $V_{\mathcal{F}}$ and $V_{\mathcal{R}}$, respectively. The TPR is defined as a tensor $T$ in a vector space $V_{\mathcal{F}} \otimes V_{\mathcal{R}}$, where $\otimes$ is the tensor product operator, which is computed as

(a) Tree structure encoded by TPR          (b) Kanerva's SDM
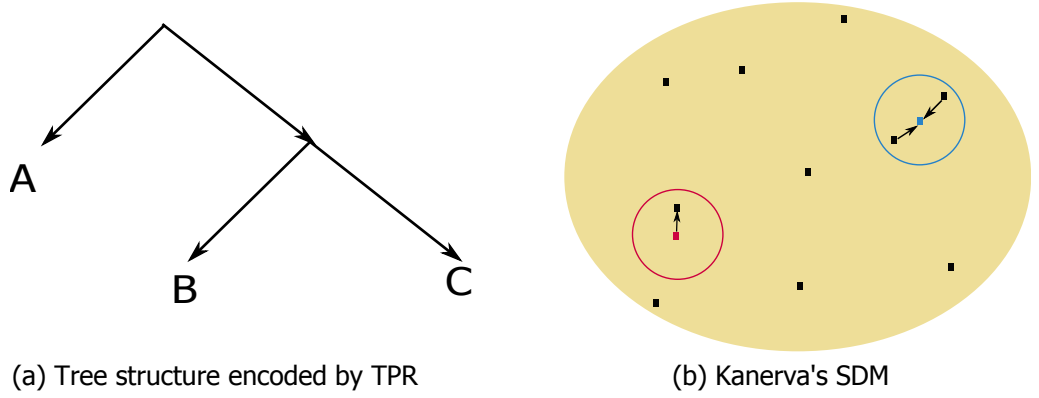
Figure 2.6: (a) Example of a tree encoded by TPR. (b) SDM's memory write (red) and read (blue) access. The read and write involve all memory locations around the queried points.

$$T = \sum_i f_i \otimes r_i \tag{2.58}$$

where $f_i$ and $r_i$ are vectors representing some filler and role, respectively. The tensor dot product $\bullet$ is used to decode the memory as follows,

$$f_j = T \bullet r_j \tag{2.59}$$

For example, the following 4 concepts have relations: $dog(bark)$ and $horse(big)$ in which the set of filler is $\mathcal{F} = \{bark, horse\}$ and the set of role is $\mathcal{R} = \{bark, big\}$. The TPR of these concepts is

$$T = f_{dog} \otimes r_{bark} + f_{horse} \otimes r_{big} \tag{2.60}$$

Or we can encode a tree structure as in Fig. 2.6 (a) by the following operations:

$$T = A \otimes r_0 \otimes + (B \otimes r_0 + C \otimes r_1) \otimes r_1 \tag{2.61}$$
$$= A \otimes r_0 \otimes + B \otimes r_0 \otimes r_1 + C \otimes r_1 \otimes r_1 \tag{2.62}$$
$$= A \otimes r_0 \otimes + B \otimes r_{01} + C \otimes r_{11} \tag{2.63}$$

This mechanism allows storing symbolic structures and grammars and thus supports reasoning. For further details, we refer readers to the original work (Smolensky, 1990) and recent application to deep learning (Schlag and Schmidhuber, 2018).

### 2.4.4   Sparse Distributed Memory

Matrix memory is a direct extension to vector memory for RNNs. There are two ways to build a matrix memory: correlation matrix memory (or tensor memory) and sparse distributed memory. While the former focuses on storing the associations amongst items (e.g., Hopfield network, Holographic memory and CMM), the latter aims to store each item as a high-dimensional vector, which is closer to Random Access Memory in computer architecture. Because each vector is physically stored in a memory slot, we also refer to this model as slot-based memory. Sparse distributed memory (SDM) can represent correlation matrix memory, computer memory, feed-forward artificial neural networks and associative-memory models of the cerebellum. Such a versatility naturally results in SDM's applications to RNN as one form of external memory.

**Kanerva memory model**

In 1988, Pentti Kanerva introduced the SDM as a new approach to model human long-term memory (Kanerva, 1988). The model revolves around a simple idea that the distances between concepts in our minds correspond to the distances between points of a high-dimensional space. As we, when hinted by key signals, tend to remember specific things such as individual, object, scene and place, the brain must make the identification nearly automatic, and high-dimensional vectors as internal representations of things do that. Another important property of high dimensional spaces is that distance between two random points should be far, which allows inexact representation of the point of interest. In other words, using long vectors to store items enables a fault-tolerant and robust memory.

The SDM stores items (binary vectors) in a large number of hard locations or memory slots whose addresses ($m_a$) are given by binary strings of length $D$, randomly distributed throughout the address space $\{0, 1\}^D$. Input to the memory consists of two binary patterns, an address pattern (location to be accessed) and a content pattern (item to be stored). The pattern is called self-addressing when its content is also its address. Furthermore, in SDM, each memory slot $m$ is armed with a vector of counters $m_c$ initialised to 0 with the same length of the content. The memory operations are based on similarity between the addresses.

---

**Algorithm 2.1** Memory writing in SDM

---
**Require:** input $x$ and SDM
 1: Find a set of chosen locations $M(x)$ using Eq. (2.64)
 2: **for each** $m$ in $M(x)$ **do**
 3:     **for** $i = 1, D$ **do**
 4:         **if** $x_c[i] == 1$ **then**
 5:             $m_c[i] += 1$
 6:         **else**
 7:             $m_c[i] -= 1$
 8:         **end if**
 9:     **end for**
10: **end for**

---

**Memory writing**   When storing input item $x = (x_a, x_c)$ to the SDM, the address pattern $x_a$ is compared against all memory location addresses. Relevant physical locations to consider are those which lie within a hypersphere of radius $r$ centered on the address pattern point

$$M(x) = \{m : d(m_a, x_a) < r\} \tag{2.64}$$

where $d$ is some similarity measure between 2 vectors. In the original model, Kanerva used Hamming distance. The content is distributed in the set of locations $M(x)$ as in Algo. 2.1.

**Memory reading**   Basically, reading from any point in the memory space pools the data of all nearby locations. Given a cue address $x'_a$, contents of the counters at locations near $x'_a$ are summed and thresholded at zero to return the binary content. The proximity criteria still follows Eq. (2.64). The reading mechanism allows SDM to retrieve data from imprecise or noisy cues. Fig. 2.6 (b) visualises the memory access behaviors.

The assumption underlying the original SDM are: (i) the location addresses are fixed, and only the contents of the locations are modifiable, (ii) the locations are sparse and distributed across the address space $\{0, 1\}^D$ (e.g., randomly sample $10^6$ addresses from an address space of 1000 dimensions ). These assumptions make the model perform well on storing random input data.

**SDM as an associative matrix memory**   We can implement SDM by using three operations of associative memory. The minimum setting for this implementa-

tion includes:

- A hard-address matrix $A \in \mathbb{B}^{N \times D}$ where $N$ and $D$ are the numbers of memory locations and the dimension of the address space, respectively.

- A counter (content) matrix $C \in \mathbb{B}^{N \times D}$.

- Cosine similarity is used to measure proximity.

- Threshold function $\boldsymbol{y}$ that maps distances to binary values:$\boldsymbol{y}(d) = 1$ if $d \geq r$ and vice versa.

- Threshold function $\boldsymbol{z}$ that converts a vector to binary vector: $\boldsymbol{z}(x) = 1$ if $x \geq 0$ and vice versa.

Then, the memory writing $(\otimes + \oplus)$ and reading $(\bullet)$ become

$$C := C + \boldsymbol{y}(Ax_a) x_c^\top \tag{2.65}$$

$$x_c' = \boldsymbol{z}\left(C^\top \boldsymbol{y}(Ax_a')\right) \tag{2.66}$$

These expressions are closely related to attention mechanisms commonly used nowadays (Sec. 3.2.2).

In general, SDM overcomes limitations of correlation matrix memory such as Hopfield network since the number of stored items in SDM is not limited by the number of processing elements. Moreover, one can design SDM to store a sequence of patterns. Readers are referred to Keeler (1988) for a detailed comparison between SDM and Hopfield network (Keeler, 1988).

**Memory-augmented neural networks and attention mechanisms**

The current wave of deep learning has leveraged the concept of SDM to external neural memory capable of supporting the working memory of RNNs (Weston et al., 2014; Graves et al., 2014, 2016; Miller et al., 2016). These models enhance the SDM with real-valued vectors and learnable parameters. For example, the matrices $A$ and $C$ can be automatically generated by a learnable neural network. To make whole architecture learnable, differentiable functions and flexible memory operations must

be used. Attention mechanisms are the most common operations used in MANNs to facilitate the similarity-based memory access of SDM. Through various ways of employing attentions, RNNs can access the external memory in the same manner as one accesses SDM. Details on neural distributed (slot-based) memory and attention mechanisms will be provided in Chapter 3.

## 2.5 Relation to Computational Models

Automatons are abstract models of machines that perform computations on an input by moving through a series of states (Sipser et al., 2006). Once the computation reaches a finish state, it accepts and possibly produces the output of that input. In terms of computational capacity, there are three major classes of automaton:

- Finite-state machine

- Pushdown automata

- Turing machine

Pushdown automata and Turing machine can be thought of as extensions of finite-state machines (FSMs) when equipped with an external storage in the form of stack and memory tape, respectively. With stored-program memory, an even more powerful class of machines, which simulates any other Turing machines, can be built as universal Turing machine (Turing, 1936). As some Turing machines are also universal, they are usually regarded as one of the most general and powerful automata besides universal Turing machines.

One major objective of automata theory is to understand how machines compute functions and measure computation power of models. For example, RNNs, if properly wired, are Turing-complete (Siegelmann and Sontag, 1995), which means they can compute arbitrary sequences if they have unlimited memory. Nevertheless, in practice, RNNs struggle to learn from the data to predict output correctly given simple input sequence (Bengio et al., 1994). This poses a question on the effective computation power of RNNs.

Another way to measure the capacity of RNNs is via simulations of operations that they are capable of doing. The relationship between RNNs and FSMs has been

discovered by many (Giles et al., 1992; Casey, 1996; Omlin and Giles, 1996; Tiňo et al., 1998), which suggest that RNNs can mimic FSMs by training with data. The states of an RNN must be grouped into partitions representing the states of the generating automation. Following this line of thinking, we can come up with neural architectures that can simulate pushdown automata, Turing machine and universal Turing machine. Neural stack is an example which arms RNN with a stack as its external memory (Mozer and Das, 1993; Joulin and Mikolov, 2015; Grefenstette et al., 2015). By simulating push and pop operations, which are controlled by the RNN, neural stack mimics the working mechanism of pushdown automata. Neural Turing Machine and its extension Differentiable Neural Computer (Graves et al., 2014, 2016) are prominent neural realisations of Turing machine. They use an RNN controller to read from and write to an external memory in a manner resembling Turing machine's operations on its memory tape. Since the memory access is not limited to the top element as in neural stack, these models have more computational flexibility. Until recently, Le et al. (2020) extended the simulation to the level of universal Turing machine by employing the stored-program principle (Turing, 1936; von Neumann, 1993). We save a thorough analysis on the correspondence between these MANNs and Turing machines for Chapter 7. Here, we briefly draw a correlation between models of recurrent neural networks and automata (see Fig. 2.7).

It should be noted that the illustration is found on the organisation of memory in the models rather than the computational capacity. For example, some Turing machines are equivalent to universal Turing machine in terms of capacity; RNNs are on par with other MANNs because they are all Turing-complete. Having said that, when neural networks are organised in a way that simulates powerful automata, their effective capacity is often greater and thus, they tend to perform better in complicated sequential learning tasks (Graves et al., 2014, 2016; Le et al., 2020). A similar taxonomy with proof of inclusion relation amongst models can be found in the literature (Ma and Principe, 2018).

## 2.6   Closing Remarks

We have briefly reviewed different kinds of memory organisations in the neural network literature. In particular, we described basic neural networks such as Feed-forward and Recurrent Neural Networks and their primary forms of memory con-
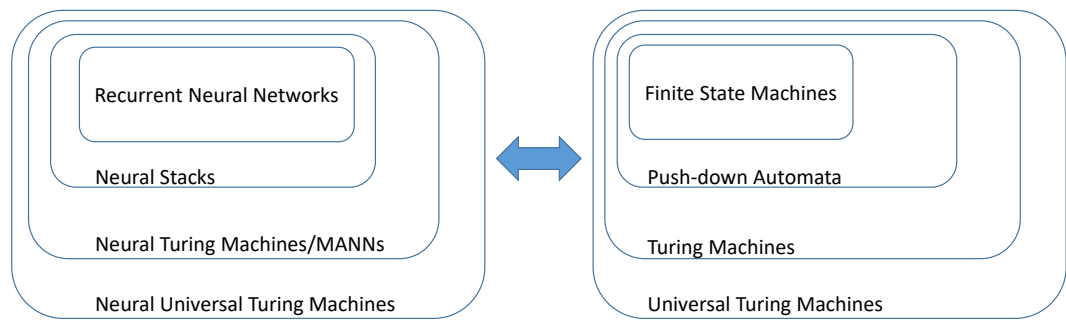
Figure 2.7: Relation between external memory and computational models

structies, followed by a taxonomy on mathematical models of well-known external memory designs based on memory operational mechanisms and relations to automation theory. In the next chapter, we narrow the scope of literature review to the main content of this thesis: Memory-augment Neural Networks and their extensions.

# Chapter 3

# Memory-augmented Neural Networks

## 3.1 Gated RNNs

### 3.1.1 Long Short-Term Memory

Despite its ability to model temporal dependencies in sequential data, RNNs face a big mathematical challenge of learning long sequences. The basic problem is that gradients propagated over many steps tend to either vanish or explode. Although the explosion can be prevented with the use of activation functions (i.e., tanh or sigmoid) that restrict the range of update values, the vanishing problem remains with these nonlinear activation functions (Sec. 2.4.1). The difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions compared to short-term ones. In practice, experiments have shown that RNNs might find it hard to learn sequences of only length 10 or 20 (Bengio et al., 1994).

Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) is introduced as a simple yet clever way to alleviate the problem. The core idea is to produce paths where the gradient can flow for long duration by adding a linear self-loop memory cell to the computation of the hidden unit. Notably, the weight of the linear self-loop is gated (controlled by another hidden unit) and dependent on the input. This enables the network to dynamically moderate the amount of
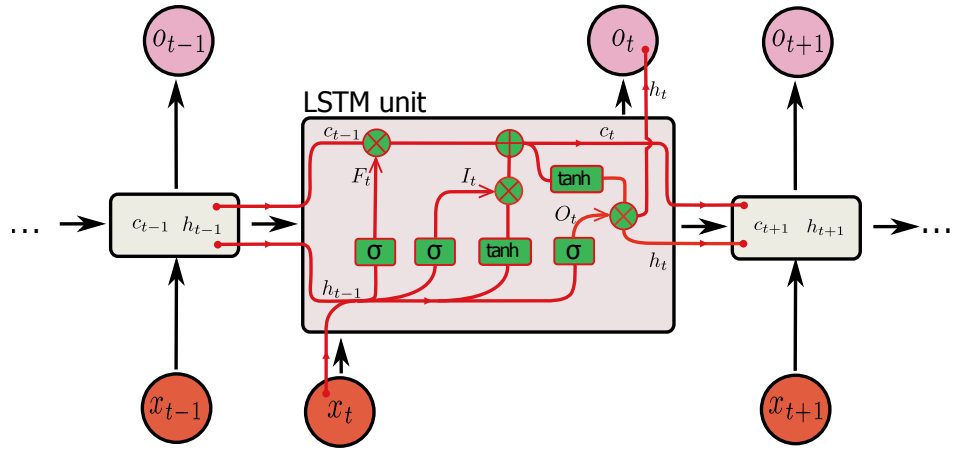
Figure 3.1: Block diagram of a modern LSTM unit. $\times$ and $+$ are element-wise product and add operators, respectively. $\sigma$ and tanh are sigmoid and tanh functions, respectively.

information passed by the hidden unit. In LSTM, there is a system of gating units that controls the flow of information, as illustrated in Fig. 3.1. The modern LSTM model is slightly different from the original LSTM presented in Sec. 2.4.1, in which we move from neuronal to vector representation with additional parameters.

The most important component is the cell memory $c_t$, which has a linear self-loop formulation

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{3.1}$$

where $f_t$ is the forget gate, $c_{t-1}$ is the previous cell value, $i_t$ is the input gate, $\tilde{c}_t$ is the candidate value for current cell memory and $*$ denotes element-wise multiplication. Similar to RNN's hidden state computation (Eq. (2.8)), $\tilde{c}_t$ is calculated as the following,

$$\tilde{c}_t = \tanh\left(W_c h_{t-1} + U_c x_t + b_c\right) \tag{3.2}$$

The gates are also functions of previous hidden state and current input with different parameters

$$f_t = \sigma \left( W_f h_{t-1} + U_f x_t + b_f \right) \tag{3.3}$$

$$i_t = \sigma \left( W_i h_{t-1} + U_i x_t + b_i \right) \tag{3.4}$$

$$o_t = \sigma \left( W_o h_{t-1} + U_o x_t + b_o \right) \tag{3.5}$$

where $\sigma$ is the sigmoid function that keeps the gate values in range $[0, 1]$. The final hidden state $h_t$ is computed based on the cell memory $c_t$, gated by the output gate $o_t$ as follows,

$$h_t = o_t * \tanh \left( c_t \right) \tag{3.6}$$

Given the hidden state $h_t$, other computations for the output $o_t$ are the same as in Elman's RNN (Eq. (2.9)).

In LSTM, the forget gate $f_t$ plays a crucial role in enabling the network to capture long-term dependencies. If $f_t \to 1$, the previous memory will be preserved and thus, the product of derivatives associated with a distant input is close to one. This allows a distant input to take part in the backpropagation update and slow down the gradient vanishing process. If $f_t \to 0$, the path to previous cells is disconnected and the model tends to remember only short-term events.

Empirical results have shown that LSTM networks learn long-term dependencies more easily than the simple RNNs. State-of-the-art performances were obtained in various challenging sequence processing tasks (Graves and Schmidhuber, 2005; Vinyals and Le, 2015). Other simpler alternatives to LSTM have been studied including Highway Networks (Srivastava et al., 2015) and GRUs (Cho et al., 2014b).

### 3.1.2 Gated Recurrent Unit

One simplified variant of LSTM is Gated Recurrent Unit (GRU) (Cho et al., 2014b), which uses two multiplicative gates to harness the vanishing gradients problem and capture longer dependencies in the sequence. Unlike LSTM, GRU does not require a separate memory cell. At each timestep, using a reset gate $r_t$, the model computes

a candidate hidden state $\tilde{h}_t$ as follows,

$$r_t = \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right) \tag{3.7}$$

$$\tilde{h}_t = \tanh\left(W_h x_t + U_h\left(r_t * h_{t-1}\right) + b_h\right) \tag{3.8}$$

The candidate hidden state is determined by current input and previous hidden state. When $r_t$ is close to 0, the candidate hidden state is reset with the current input, allowing the model to delete any irrelevant information from the past. The hidden state is then updated by linear interpolation between the previous hidden state and the candidate hidden state

$$h_t = z_t * h_{t-1} + (1 - z_t) * \tilde{h}_t \tag{3.9}$$

where an update gate $z_t$ decides how much the hidden state should update its content. The removal of input gate prevents the amount of information in the hidden states from exploding. $z_t$ is computed by

$$z_t = \sigma\left(W_z x_t + U_z h_{t-1} + b_z\right) \tag{3.10}$$

A main advantage of GRU compared with LSTM is that GRU can run faster while maintaining comparable performance (Chung et al., 2014). The reduction of parameters also helps GRU less overfit to training data as LSTM does.

## 3.2 Attentional RNNs

### 3.2.1 Encoder-Decoder Architecture

Intuitively, attention mechanism is motivated by human visual attention where our eyes are able to focus on a certain region of an image/language with "high resolution" while perceiving the surrounding context in "low resolution". This focus is adjusted dynamically overtime and directly contributes to our decision making process. Before going into details, we will briefly review sequence-to-sequence model–a recurrent architecture that is often used with attention mechanism.

(a) Seq2Seq     (b) Seq2Seq with attention mechanism
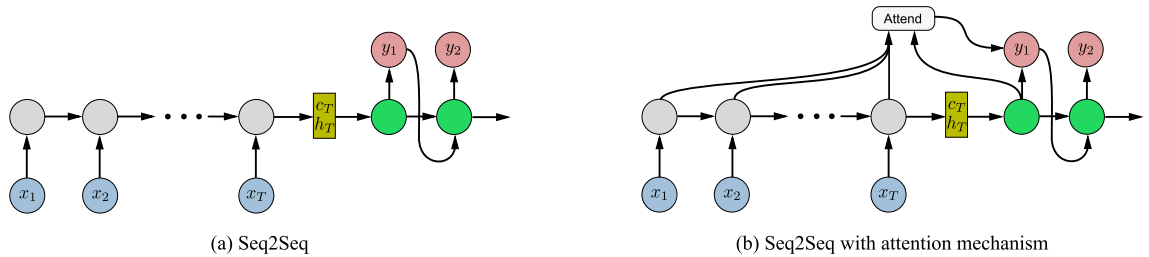
Figure 3.2: (a) Seq2Seq Model. Gray and green denote the LSTM encoder and decoder, respectively. In this architecture, the output at each decoding step can be fed as input for the next decoding step. (b) Seq2Seq Model with attention mechanism. The attention computation is repeated across decoding steps.

Amongst sequential modeling tasks, sequence-to-sequence mapping is one of the most challenging one whose practical applications may include machine translation, document summarisation and dialog response generation. To solve such tasks, we may use an RNN-like encoder to model the input sequence and then an RNN-like decoder to model the output sequence. To link the two models, the final hidden state of the encoder (thought vector) is passed to the decoder as the latter's initial hidden state (see Fig. 3.2 (a)). This encoder-decoder architecture, often referred to as Seq2Seq, is firstly introduced by Cho et al. (2014) and has demonstrated superior performance over LSTM in machine translation (Cho et al., 2014a; Sutskever et al., 2014b).

### 3.2.2 Attention Mechanism

Even when applying LSTM to Seq2Seq helps to ease the gradient vanishing in general, the decoder in Seq2Seq is likely to face this problem when the number of decoding steps becomes larger. Given that the decoder receives a fixed-size though vector representing the whole input sequence, it is hard to recover the contribution of distant encoding input in predicting decoder's outputs. To overcome this, Bahdanau et al. (2015) proposed using attention mechanism in encoder-decoder architecture. The key idea is to let the decoder look over every piece of information that the original input sequence holds at every decoding step, which is equivalent to creating a direct connection from a decoder unit to any encoder unit (see Fig. 3.2 (b)). Each connection then will be weighted by an attention score, which is a function of hidden states from both encoder and decoder. The weight $\alpha_{ij}$ between the $i$-th decoding step and the $j$-th encoding step is defined as

$$e_{ij} = v^T \tanh\left(W s_{i-1} + U h_j\right) \tag{3.11}$$

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum\limits_{k=1}^{L} \exp\left(e_{ik}\right)} \tag{3.12}$$

where $e_{ij}$ is the unnormalised weight, $v$ is a parametric vector and $W$, $U$ are parametric matrices. $s$ and $h$ are used to denote the hidden state of the decoder and the encoder, respectively. Eq. (3.12) is the well-known softmax function to make the weights sum to one over $L$ encoding steps. Then, a context vector for the $i$-th decoding step is computed using a weighted summation of all encoder's hidden states as follows,

$$c_i = \sum_{j=1}^{L} \alpha_{ij} h_j \tag{3.13}$$

Finally, the context vector $c_i$ is combined with the decoder hidden state $s_i$ to compute the $i$-th decoder's output and next state (Bahdanau et al., 2015). Attention mechanism has several modifications such as hard attention (Xu et al., 2015) and pointer network (Vinyals et al., 2015).

### 3.2.3 Multi-Head Attention

Traditional RNNs read a sequence step by step to extract sequential dependencies, which is slow and hard to capture far apart relations. Attention helps link two distant timesteps quickly and thus, shows potential to replace completely RNNs in modeling sequential data. However, the vanilla attention mechanism is shallow with one step of computation per timestep, which relies on the hidden state of RNNs for richer representation. In an effort to replace RNNs with attention, Vaswani et al. (2017) proposed a deeper attention mechanism with multiple heads implemented efficiently using dot-product operation. The model reads all timesteps in the sequence at once like Feed-forward Neural Networks, which utilises parallel computing. Moreover, multiple keys, values and queries are packed into matrices $K$, $V$ and $Q$, respectively. Then, the multi-head attention operation is computed as follows,

$$Attention\,(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V \qquad (3.14)$$

where $d_k$ is the number of key dimension. The multi-head attention lies at the core of self-attention mechanism, in which, relational features are encoded from the input sequence (Fig. 3.3 (a)). Similarly, the output sequence features can be extracted and combined with the encoded input to form an encoder-decoder architecture called The Transformer. (Fig. 3.3 (b)).

The Transformer has empirically demonstrated that attention alone can replace recurrent models in solving sequential tasks including machine translation and language parsing (Vaswani et al., 2017). This opens a new research direction in deep learning where attention can be used to extract relations between time-ordered events. The limitation of self-attention is its quadratic complexity. However, this can be compensated with parallel computation ability. Detailed discussion of this new research angle is beyond the scope of this thesis. Instead, we will focus on slot-based memory networks, another approach with attention that is built upon a readable/writable external memory. The approach resembles closely SDM as well as human associative memory.

## 3.3  Slot-Based Memory Networks

### 3.3.1  Neural Stack

Traditional stack is a storage of elements that works on the principle of last-in-first-out, which describes the order in which the elements come off a stack. In general, stack supports two operations: push, which adds an element to the stack, and pop, which removes the most recently added element (the top one). Additionally, a peek operation may give access to the value of the top element without modifying the stack. Stack is a convenient memory for solving problems with hierarchical structures because it stores the temporary results in a way that supports backtracking and tree traversal. Recently, researchers have tried to implement continuously differentiable prototype of traditional stacks using deep networks (Joulin and Mikolov, 2015; Grefenstette et al., 2015). We briefly review the implementations proposed by Grefenstette et al. (2015) that aim to mimic Stack, Queue and Deque on solving natural language transduction problems.

(a) Self-attention                     (b) The Transformer
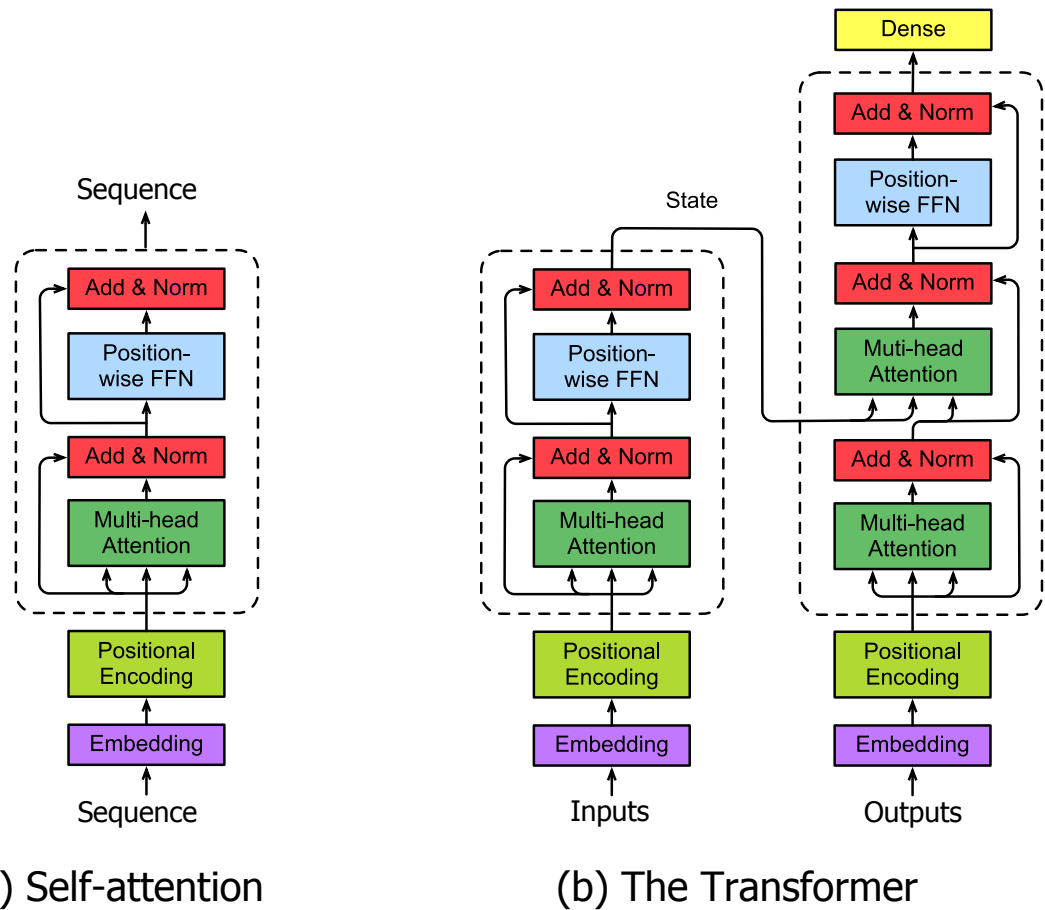
Figure 3.3: Computation stages of the encoding using self-attention (a) and encoding-decoding architecture–The Transformer (b). Embedding layers convert input/output tokens to vectors of fix dimension, followed by Positional Encoding layers that add temporal information to each vector. The main block of computation combines multi-head attention, residual connection, layer normalisation and Feed-forward layers, which can be repeated multiple times.

In the implementations, a row-expandable matrix $V$ is used to store the data. The $i$-th row $V[i]$ is associated with a strength scalar $s[i]$. When $v_t$–the item at timestep $t$– is presented, its value is added to the matrix $V$ and never be modified, which yields,

$$V_t[i] = \begin{cases} V_{t-1}[i] = v_i & \text{if } 1 \leq i < t \\ v_t & \text{if } i = t \end{cases} \tag{3.15}$$

To modify the stack content under push and pop operations, we modify the strength vector instead as the following,

$$s_t[i] = \begin{cases} \max\left(0, s_{t-1}[i] - \max\left(0, u_t - \sum_{j=i+1}^{t-1} s_{t-1}[j]\right)\right) & \text{if } 1 \leq i < t \\ d_t & \text{if } i = t \end{cases} \tag{3.16}$$

where $u_t$ and $d_t$ are the pop and push signals generated by a neural network, respectively. Basically, the strength for the top item is set to the push signal. Then, we want to subtract the strength of stored items ($s_t[i]$) by an amount of the pop signal ($u_t$) from the top (highest index) to the bottom (lowest index) of the stack. If the pop signal is greater than the strength, the strength of the item is set to 0 (totally popped out of the stack) and the remainder of the pop signal is passed to lower items until we run out of pop signal. The peek or read operation is carried out by

$$r_t = \sum_{i=1}^{t} \left( \min\left( s_t[i], \max\left( 0, 1 - \sum_{j=i+1}^{t} s_t[j] \right) \right) \right) V_t[i] \tag{3.17}$$

The output $r_t$ of the read operation is the weighted sum of the rows of $V_t$, scaled by the temporary strength values created during the traversal. Intuitively, items with zero strength do not contribute to read value and items on the bottom contribute less than those near the top. Neural Queue and DeQue can be implemented in similar manners by modifying Eqs. (3.15)-(3.17).

A controller implemented as RNN is employed to control stack operations. The current input $i_t$ from the sequence and the previous read-out $r_{t-1}$ will be concatenated as input for the RNN to produce the current hidden state $h_t$ and the controller output $o'_t$. The controller output will be used to generate the item, control signals and final output of the whole network as follows,

$$d_t = \sigma\left(W_d o_t' + b_d\right) \tag{3.18}$$

$$u_t = \sigma\left(W_u o_t' + b_u\right) \tag{3.19}$$

$$v_t = \tanh\left(W_v o_t' + b_v\right) \tag{3.20}$$

$$o_t = \tanh\left(W_o o_t' + b_o\right) \tag{3.21}$$

Experiments have demonstrated that the proposed models are capable of solving transduction tasks for which LSTM-based models falter (Grefenstette et al., 2015).

### 3.3.2 Memory Networks

One solution to ensure a model will not forget is to create a slot-based memory module and store every piece of information into the memory slots. The memory can be implemented as a matrix $M \in \mathbb{R}^{N \times D}$ whose rows contain vectors representing the considering piece of information. Here, $N$ is the number of slots and $D$ is the dimension of the representation vector (word size). Following this principle, Memory Network (MemNN) (Weston et al., 2014) stores all information (e.g., knowledge base or background context) into an external memory. When there is a retrieval request, it assigns a relevance probability to each memory slot using content-based attention scheme, and reads contents from each memory slot by taking their weighted sum. Since the model is designed for language understanding, each slot of the memory often associates with a document or a sentence. When a query/question about facts related to the stored documents is presented, MemNN will perform content-based attention as follows,

$$p_i = \text{softmax}\left(u^T m_i\right) \tag{3.22}$$

where $u$ is the feature and $m_i$ is the memory's $i$-th row vector, which represent the query and the stored document, respectively. $p_i$ is the attention score to the $i$-th memory slot, normalised by softmax function.

The output of the memory, given query $u$, is the read vector

$$r = \sum_{i=1}^{N} p_i c_i \tag{3.23}$$

where $c_i$ is the output vector corresponding to the $i$-th slot. In MemNN, it is trainable parameter while in key-value memory network (Miller et al., 2016), it comes from the data. Then, the model can make prediction by feeding the read values to another feed-forward neural network.

A multi-hop version MemN2N has also been studied and outperforms LSTM and MemNN in question-answering tasks (Sukhbaatar et al., 2015). MemN2N extends MemNN by adding refinement updates on the query and the read-out. The refinement reads

$$u_{k+1} = Hu_k + r_k \tag{3.24}$$

where $H$ is a parametric matrix and $k$ is the refinement step.

Although memory networks have big advantages over LSTM due to the use of external matrix memory, it is hard to scale to big dataset since the number of memory slots grows linearly with the number of data. Some tricks such as hashing have been proposed but they have a trade-off between capacity and accuracy. More importantly, it is unlikely that we tend to store everything in our brain. We have the ability to forget the old memory and update with new knowledge, which is ignored by memory network designs.

### 3.3.3 Neural Turing Machine

In contrast to MemNN, Neural Turing Machine (NTM) (Graves et al., 2014) introduces a slot-based read/write mechanism to the memory module. The memory size does not need to equal the number of considering pieces of information. The model learns to overwrite obsolete or unimportant memory slots with recent and useful information to optimise a final goal. This writing scheme fits with sequential task where the prediction goal can be achieved without paying attention to all timestep inputs. To control the memory operations, NTM uses a neural controller network whose parameters are slow-learning weights. The controller is responsible for determining instantly after each timestep the content to be read from and written to the memory. An illustration of NTM components is described in Fig. 3.4 (a).

In NTM, both reading and writing locations are determined by the address, which is a weight over the memory slots. The weight is initially computed by the content-

(a) NTM

(b) DNC
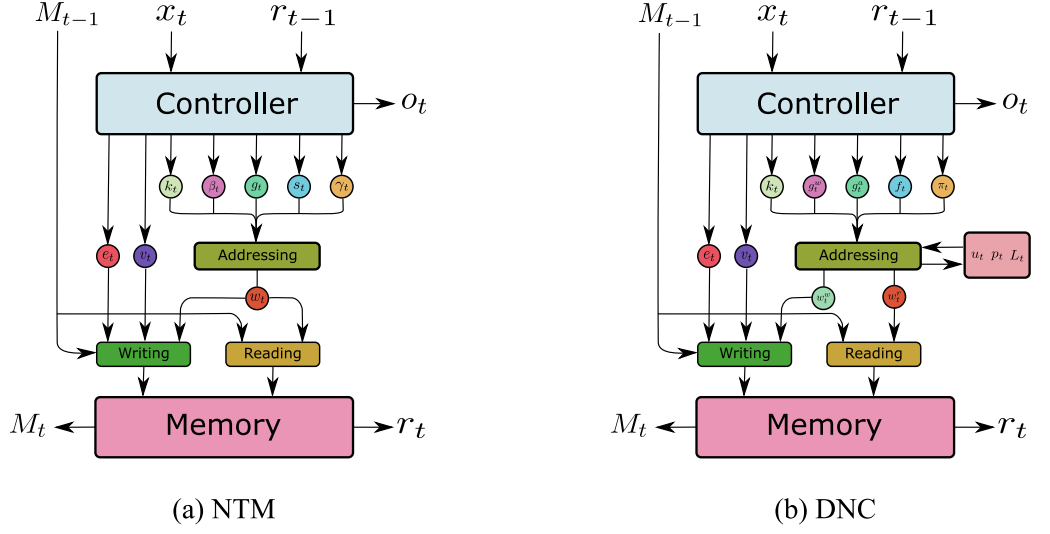
Figure 3.4: (a) Architecture of NTM. Circles denote intermediate variables computed by the controller. The controller takes the current timestep data $x_t$ and the previous read value $r_{t-1}$ as the input and produces $r_t$, updates memory $M_t$ and predict output $o_t$. (b) Architecture of DNC. The operation is similar to NTM's with extra modules to keep track of memory usage $u_t$, precedence $p_t$ and link matrix $L_t$.

based attention,

$$w_t^c(i) = \frac{\exp\left(\beta_t m\left(k_t, M_t(i)\right)\right)}{\sum\limits_{j=1}^{D} \exp\left(\beta_t m\left(k_t, M_t(j)\right)\right)} \tag{3.25}$$

Here, $w_t^c \in \mathbb{R}^N$ is the content-based weight, $\beta_t$ is a strength scalar, $m$ is a matching function that measures the similarity between a key $k_t \in \mathbb{R}^D$ and the $i$-th memory slot $M_t(i)$. In practice, $m$ is implemented as cosine similarity

$$m\left(k_t, M_t(i)\right) = \frac{k_t \cdot M_t(i)}{||k_t|| \cdot ||M_t(i)||} \tag{3.26}$$

Besides the content-based addressing, NTM supports location-based addressing started with an interpolation between content-based weight and the previous weight

$$w_t^g = g_t w_t^c + (1 - g_t) w_t \tag{3.27}$$

where $g_t$ is the interpolation gate. This allows the system to learn when to use (or ignore) content-based addressing. Also, the model is able to shift focus to other rows by performing convolution shift modulo $R$ as the following,

$$\tilde{w}_t(i) = \sum_{j=0}^{R} w_t^g(i) s_t(i-j) \tag{3.28}$$

where $s_t$ is the shift weighting. Finally, sharpening is used to prevent the shifted weight from blurring, which results in the final weight

$$w_t(i) = \frac{\tilde{w}_t(i)^\gamma}{\sum\limits_{j} \tilde{w}_t(j)^\gamma} \tag{3.29}$$

Given the weight calculated, the memory update is defined by using these bellowing equations

$$M_t^{erased}(i) = M_{t-1}(i)\left[1 - w_t(i) e_t\right] \tag{3.30}$$

$$M_t(i) = M_t^{erased}(i) + w_t(i) v_t \tag{3.31}$$

where $e_t \in \mathbb{R}^D$ and $v_t \in \mathbb{R}^D$ are erase vector and update vector, respectively. The read value is computed using the same address weight as follows,

$$r = \sum_{i=1}^{N} w_t(i) M_t(i) \tag{3.32}$$

The controller can be implemented as a feed-forward network or LSTM fed with an concatenation of the read-out $r_t$ and the timestep data $x_t$. The computation of the output $o_t$ follows the same computing mechanism of the controller network (see Sec. 3.1.1).

With a fixed size external memory, NTM can scale well when dealing with very long sequence while maintaining better remembering capacity than other recurrent networks such as RNN, GRU and LSTM. Experiments have shown NTM outperforms LSTM by a huge margin in memorisation testbeds including copy, repeat copy, associative recall and priority sort (Graves et al., 2014).

### 3.3.4  Differentiable Neural Computer

In this subsection, we briefly review DNC (Graves et al., 2016), a powerful extension of the NTM. A DNC consists of a controller, which accesses and modifies an external memory module using a number of read heads and one write head. Given some input $x_t$, and a set of $R$ read values from memory $r_{t-1} = \left[ r_{t-1}^1, ..., r_{t-1}^k, ..., r_{t-1}^R \right]$, the controller produces the output $o_t$ and the interface which consists of intermediate variables, as depicted in Fig. 3.4 (b). DNC also uses the content-based attention in Eq. (3.25) to determine the content-based write-weight $w_t^{cw}$ and read-weights $w_t^{cr,k}$. However, different from NTM, DNC does not support location-based attention. Instead, DNC introduces dynamic memory allocation and temporal memory linkage for computing the final write-weight $w_t^w$ and read-weights $w_t^{r,k}$ separately.

**Dynamic memory allocation & write weightings:** DNC maintains a differentiable free list tracking the usage $u_t \in [0,1]^N$ for each memory location. Usage is increased after a write and optionally decreased after a read, determined by the free gates $f_t^k$ as follows,

$$u_t = \left( u_{t-1} + w_{t-1}^w - u_{t-1} \circ w_{t-1}^w \right) \prod_{k=1}^R \left( 1 - f_t^k w_t^{r,k} \right) \qquad (3.33)$$

The usage is sorted and then the allocation write-weight is defined as

$$a_t \left[ \Phi_t \left[ j \right] \right] = \left( 1 - u_t \left[ \Phi_t \left[ j \right] \right] \right) \prod_{i=1}^{j-1} u_t \left[ \Phi_t \left[ i \right] \right] \qquad (3.34)$$

in which, $\Phi_t$ contains elements from $u_t$ sorted by ascending order from least to most used. Given the write gate $g_t^w$ and allocation gate $g_t^a$, the final write-weight then can be computed by interpolating between the content-based write-weight and the allocation write-weight,

$$w_t^w = g_t^w \left[ g_t^a a_t + \left( 1 - g_t^a \right) w_t^{cw} \right] \qquad (3.35)$$

Then, the memory is updated by the following rule

$$M_t = M_{t-1} \circ \left( E - g_t^w w_t^w e_t^\top \right) + g_t^w w_t^w v_t^\top \qquad (3.36)$$

**Temporal memory linkage & read weightings:** DNC uses a temporal link matrix $L_t \in [0,1]^{N \times N}$ to keep track of consecutively modified memory locations, and $L_t[i,j]$ represents the degree to which location $i$ was the location written to after location $j$. Each time a memory location is modified, the link matrix is updated to remove old links to and from that location, and add new links from the last-written location. To compute the link matrix, DNC maintains a precedence weighting to keep track of which locations were most recently written by using the following equation

$$p_t = \left( 1 - \sum_i^N w_t^w(i) \right) p_{t-1} + w_t^w \tag{3.37}$$

Then $p_t$ is used to update the link matrix as follows,

$$L_t(i,j) = (1 - w_t^w(i) - w_t^w(j)) L_{t-1}(i,j) + w_t^w(i) p_{t-1}(j) \tag{3.38}$$

Given the link matrix, the final read-weight is given as follow,

$$w_t^{r,k} = \pi_t^k(1) L_t^\top w_{t-1}^{r,k} + \pi_t^k(2) w_t^{cr,k} + \pi_t^k(3) L_t w_{t-1}^{r,k} \tag{3.39}$$

The read mode weight $\pi_t^k$ is used to balance between the content-based read-weight and the forward $L_t w_{t-1}^{r,k}$ and backward $L_t^\top w_{t-1}^{r,k}$ of the previous read. Then, the $k$-th read value $r_t^k$ is retrieved using the final read-weight vector

$$r_t^k = \sum_i^N w_t^{r,k}(i) M_t(i) \tag{3.40}$$

The performance of DNC is better than that of NTM, LSTM and other variants of memory networks. It achieves state-of-the-art results under various experimental settings such as copy/recall, question-answering and graph reasoning (Graves et al., 2016).

### 3.3.5 Memory-augmented Encoder-Decoder Architecture

A memory-augmented encoder-decoder (MAED) consists of two neural controllers linked via external memory. This is a natural extension to read-write MANNs to

handle sequence-to-sequence problems, which has been investigated in Sec. 3.2. MAED has recently demonstrated promising results in machine translation (Britz et al., 2017; Wang et al., 2016) and healthcare (Le et al., 2018c,b; Prakash et al., 2017). In this thesis, MAED is the basic framework on which some of our proposed architectures are built upon. Here, we briefly analyse the operations of MAED. More details can be found in Chapter 4.

As mentioned in Sec. 3.2.1, Seq2Seq only allows information transfer from the encoder to the decoder via "thought vector". Attention mechanism creates shortcut connections via directly attending to other timesteps (see Sec. 3.2.2 and 3.2.3). Unlike these approaches, MAED uses an external memory as a reservoir containing past information to which the controllers attend. That is, instead of directly attending to other timesteps, the controllers (both encoder and decoder) attend to the memory to store and retrieve relevant information The information then participates in other decision-making processes such as output predictions, feature extractions and memory operations. If the number of memory slots is finite, the method keeps the computation resource linear to the length of the sequence, which is feasible for lifelong learning. Furthermore, selectively attending to a fixed number of memory slots requires the model to learn to compress the information to the external memory. On one hand, it may make the learning harder since the controllers must learn to write timestep information to the memory efficiently. On the other hand, attending to all timesteps is biologically implausible and information compression is a powerful skill that mimics the capacity of memory in the brain. Hence, this thesis has chosen MAED as the key framework to study and develop several memory and attention techniques for neural networks (Chapter 4 and 5).

## 3.4 Closing Remarks

We have reviewed MANNs, a family of recurrent neural networks with external memory, which is useful for a wide range of sequential learning tasks. Armed with operations such as gating and attention mechanisms, MANNs are able to control and update external vector or matrix memory module. The power of MANNs has been verified in learning long sequential data, in which their performances are superior to that of vanilla RNNs.

We hypothesise that MANNs are also effective in open challenges in multi-process/multi-view data, data with uncertainty, ultra-long sequences and problems required univer-

sal computation. We address these challenges in Chapters 4, 5, 6 and 7, respectively. A recurring theme across all of our proposed MANNs is found on slot-based memory architecture.

# Chapter 4

# Memory Models for Multiple Processes

## 4.1 Introduction

### 4.1.1 Multi-Process Learning

Traditional sequential learning focuses on modeling single processes, in which the sequence of outputs shares the same domain with the input sequence. We can extend the problem to broader scenarios where input and output sequences are from different processes (sequence to sequence) or there are multiple sequences acting as inputs and outputs (multi-view sequential mapping). For example, in healthcare setting, there are at least three processes that are executed: the disease progression, the treatment protocols, and the recording rules.

Let us start with a generic formulation of the multi-process learning. Let $S^{i_1}, ..., S^{i_M}$ and $S^{o_1}, ..., S^{o_N}$ denote $M$ input and $N$ output view spaces, respectively. Each sample of the multi-process problem $\left( \{X^{i_k}\}_{k=1}^M, \{Y^{o_k}\}_{k=1}^N \right)$ consists of $M$ input views: $X^{i_1} = \left\{ x_1^{i_1}, ..., x_{t_1}^{i_1}, ..., x_{L^{i_1}}^{i_1} \right\}, ..., X^{i_M} = \left\{ x_1^{i_M}, ..., x_{t_M}^{i_M}, ..., x_{L^{i_M}}^{i_M} \right\}$ and $N$ output views $Y^{o_1} = \{y_1^{o_1}, ..., y_{t_1}^{o_1}, ..., y_{L^{o_1}}^{o_1}\}, ..., Y^{o_N} = \left\{ y_1^{o_N}, ..., y_{t_N}^{o_N}, ..., y_{L^{o_N}}^{o_N} \right\}$. Each view has a particular length ($L^{i_1}, ..., L^{i_M}$ and $L^{o_1}, ..., L^{o_N}$) and can be treated as a set/sequence of events that belongs to different spaces ($x_{t_1}^{i_1} \in S^{i_1}, ..., x_{t_M}^{i_M} \in S^{i_M}, y_{t_1}^{o_1} \in S^{o_1}, ..., y_{t_N}^{o_N} \in S^{o_N}$). Each event then can be represented by an one-hot vector $v \in [0, 1]^{\|C\|}$, where $C$ can be $S^{i_1}, ..., S^{i_M}$ or $S^{o_1}, ..., S^{o_N}$. In single process, $M = N = 1$ and

$S^{i_1} = S^{o_1}$. Practical problems belonging to this setting can be solved efficiently using RNNs or LSTM as in language modeling (Mikolov et al., 2010), speech and optical character recognition (Graves et al., 2013; Graves, 2013; Graves and Schmidhuber, 2005). We focus more on other complicated cases, which are dual processes (sequence to sequence) and dual-view mapping, where $M = N = 1$, $S^{i_1} \neq S^{o_1}$ and $M = 2, N = 1$, $S^{i_1} \neq S^{i_2} \neq S^{o_1}$, respectively. These cases represent two classes of problems that traditional RNNs may struggle to deal with. Thus, we aim to solve them using MAED (Sec. 3.3.5) by proposing: (i) separate controls for two separate sub-processes (encoding, decoding) and (ii) multiple memories for multiple parallel, asynchronous processes. To account for these settings, we need more powerful MAEDs to handle the complexity of long-term dependencies and view interactions, which will be presented in the next sections using healthcare as a practical domain.

### 4.1.2 Real-World Motivation

A main motivation for our work in this chapter is modeling healthcare processes. In healthcare, a hospital visit is documented as one admission record consisting of diagnosis and treatment codes for the admission. The collection of these records are electronic medical records (EMRs) of a patient. Recently, using EMRs as the data for scientists to analyse and make treatment predictions has become the key for improving healthcare (Williams, 2008). A typical EMR contains information about a sequence of admissions for a patient and a wide range of information can be stored in each admission, such as detailed records of symptoms, data from monitoring devices, clinicians' observations. Diagnoses, procedures or drug prescriptions are the most important information stored in EMRs and are typically coded in standardised formats, each of which represents a medical process in EMR data.

In particular, diagnoses are coded using WHO's ICD (International Classification of Diseases) coding schemes. For example, in the ICD10 scheme, E10 encodes Type 1 diabetes mellitus, E11 encodes Type 2 diabetes mellitus and F32 indicates depressive episode. The treatment can be procedure or drug. The procedures are typically coded in CPT (Current Procedural Terminology) or ICHI (International Classification of Health Interventions) schemes. The drugs are often coded in ATC (Anatomical Therapeutic Chemical) or NDC (National Drug Code).

It is important to note that there are order dependencies amongst medical processes such as the diagnosis codes as well as procedure or drug codes. For example, diag-

nosis codes are often sequenced under some strict rules such as: (i) condition codes must be sequenced first, followed by the manifestation codes, (ii) primary diagnosis that describe the nature of the sequela must be coded before the secondary diagnosis that describes the original injury, (iii) single conditions that require more than one code have clear instructions that indicate which must be coded first. Similarly, procedure or drug codes often follow specific order, often corresponding to the order of diagnosis codes or the order of prescriptions. Moreover, the dependency in EMR clinical codes can be very long-term. Due to the fact that EMR data is temporally sequenced by patient medical visits, clinical codes at current admission may be related to other codes appearing in previous admissions. Since it is normal for patients to periodically visit hospital for regular health check, the number of admissions for some person should be very large and thus the dependency amongst clinical codes should be very long. Another characteristic of EMR data is its rarity. Although the number of EMR records are increasing day by day, it cannot cover many rarely-seen symptoms and treatments which appear sparsely throughout the history of EMR documentation. More seriously, rare diseases are deadly and rare treatments are expensive, so it is compulsory to make use of these rare information in order to make significant predictions.

Unfortunately, it is not easy for prediction models to capture the long-term dependencies and rarity of EMR data. Recent researches dealing with medical prediction have largely focused on modeling the admission's diagnosis and treatments as two set of codes and only capture sequential dependencies from one admission to another (Nguyen et al., 2017; Pham et al., 2017). Also, most of these methods avoid using rare data by only keeping codes that appear frequently. This approach exposes three limitations. First, using set representation ignores the internal internal sequential dependencies and thus fail to discover sequential relations amongst codes from the same admission. Second, refusing to use rare clinical events makes the contribution less significant because in healthcare, rare events are the more important ones. Third, outputs of this approach are often set of codes, which again detaches from realistic need where treatments and diagnoses have to follow strict orders. These challenges motivate new approaches that treat EMR history as sequences of correlated processes.

**Sequence to sequence mapping**

Treatment recommendation can be cast as sequence to sequence problem. Once diagnoses are confirmed, we want to predict the output sequence (treatment codes of the current visit) given the input sequence (all diagnoses followed by treatments from the first visit to the previous visit plus the diagnoses of the current visit). More formally, we denote all the unique medical codes (diagnosis, procedures and drugs) from the EMR data as $c_1, c_2, ..c_{|C|} \in C$, where $|C|$ is the number of unique medical codes. A patient's $n$-th admission's input is represented by a sequence of codes:

$$\left[ c_{d_1}^1, c_{d_2}^1, ..., ‰, c_{p_1}^1, c_{p_2}^1 ..., \varnothing, ..., c_{d_1}^{n-1}, c_{d_2}^{n-1}, ..., ‰, c_{p_1,}^{n-1}, c_{p_2}^{n-1}, ...., \varnothing, c_{d_1}^n, c_{d_2}^n, ..., ‰ \right]$$

(4.1)

Here, $c_{d_j}^k$ and $c_{p_j}^k$ are the $j$-th diagnosis and treatment code of the $k$-th admission, respectively. ‰ , $\varnothing$ are special characters that informs the model about the change from diagnosis to treatment codes and the end of an admission, respectively. This reflects the natural structure of a medical history, which is a sequence of clinical visits, each of which typically includes a subsequence of diagnoses, and a subset of treatments. A diagnosis subsequence usually starts with the primary condition followed by secondary conditions. In a subset of treatments, the order is not strictly enforced, but it may reflect the coding practice. The output of the patient's $n$-th admission is : $\left[ c_{p_1}^n, c_{p_2}^n, ..., c_{p_{L_{out}}}^n, \varnothing \right]$, in which $L_{out}$ is the length of the treatment sequence we want to predict and $\varnothing$ is used to inform the model to stop predicting. Finally, each code is represented by one-hot vector $v_c \in [0, 1]^{\|C\|}$, where $v_c = [0, ..., 0, 1, 0.., 0]$ ($v_c[i] = 1$ if and only if $v_c$ represents $c_i$). Unlike set encoding of each admission, representing the data in this way preserves the admission's internal order information allowing sequence-based methods to demonstrate their power of capturing sequential events.

In Sec. 4.3, we present a novel treatment recommendation model using memory network to remember long-term dependencies and rare events from EMR data. This model is built upon Differential Neural Computer (DNC) (Graves et al., 2016), a fully differentiable implementation of memory-augmented neural network (MANN) (see Sec. 3.3.4). In question-answer bAbI task (Weston et al., 2015), DNC treats the story, question as sequences of words and perform well on the task of predicting sequence of answers, which suggests the power of DNC in handling sequence input and solving sequence prediction task. Despite of its successes, DNC have never been applied to realistic domain such as healthcare, especially in clinical treatment sequence prediction. This realisation motivates us to design a DNC-based architecture that fits and works well with healthcare domain. In our design, we make use of two

controllers instead of one to handle dual processes: diagnoses and treatments. Each controller will employ different remembering strategies for each process and thus increase the robustness of prediction and the speed of learning. Besides, we apply a write-protected policy for our controller to direct the model toward reasonable remember strategies.

**Two-view sequential learning**

For two-view problems, the generic notations simplify to $S^{i_1}$, $S^{i_2}$ denoting input view spaces and $S$ the output view space. Each sample of the two-view problem $(X^{i_1}, X^{i_2}, Y)$ consists of two input views: $X^{i_1} = \left\{ x_1^{i_1}, ..., x_{t_1}^{i_1}, ..., x_{L^{i_i}}^{i_1} \right\}$, $X^{i_2} = \left\{ x_1^{i_2}, ..., x_{t_2}^{i_2}, ..., x_{L^{i_2}}^{i_2} \right\}$ and one output view $Y = \{ y_1, ..., y_t, ..., y_L \}$. Each view has a particular length ($L^{i_1}$, $L^{i_2}$ or $L$), each of which is a set/sequence of events that belongs to different spaces ($x_{t_1}^{i_1} \in S^{i_1}$, $x_{t_2}^{i_2} \in S^{i_2}$, $y_t \in S$). It should be noted that this formulation can be applied to many situations including video-audio understanding, image-captioning and other two-channel time-series signals. Here we focus effort on solving the two-view problems in healthcare.

For example, in drug prescription, doctors prescribe drugs after considering diagnoses and procedures administered to patients. In modeling disease progression, doctor may refer to patient's history of admissions to help diagnoses the current diseases or to predict the future disease occurrences of the patient. There are clinical recording rules applying to EMR codes such that diagnoses are "ordered by priority" or procedures follow the order that "the procedures were performed"[1]. Besides, although medical codes from different views are highly correlated, they are not aligned. For instances, some diagnoses may correspond to one procedure or one diagnosis may result in multiple medicines. Hence, these problems can be treated as asynchronous two-view sequential learning.

In the drug prescription context, $S^{i_1}$ and $S^{i_2}$ represent the diagnosis and procedure spaces, respectively and $S$ corresponds to the medicine space. The drug prescription objective is to select an optimal subset of medications from $S$ based on diagnosis and procedure codes. Similarly, we can formulate the disease progression problem as two input sequences (diagnoses and interventions) and one output set (next diagnoses). Although our architecture can model sequential output, the choice of representing output as set is to follow a common practice in healthcare where the order of medical

---

[1]https://mimic.physionet.org/mimictables/

suggestions is specified. Because a patient may have multiple admission records for different hospital visits, a patient record can be represented as $\{(X_a^{i_1}, X_a^{i_2}, Y_a)\}_{a=1}^{A}$, where $A$ is the number of admissions this patient commits. In order to predict $Y_a$, we may need to exploit not only $(X_a^{i_1}, X_a^{i_2})$ but also $\left\{\left(X_{pa}^{i_1}, X_{pa}^{i_2}\right)\right\}_{pa=1}^{a-1}$. More details on how our work makes use of previous admissions and handles long-term dependencies will be given in Sec. 4.4.3.

In Sec. 4.4, we propose a novel memory augmented neural network model solving the problem of asynchronous interactions and long-term dependencies at the same time. Our model makes use of three neural controllers and two external memories constituting a dual memory neural computer. In our architecture, each input view is assigned to a controller and a memory to model the intra-view interactions in that particular view. At each time step, the controller reads an input event, updates the memory, and generates an output based on its current hidden state and read vectors from the memory. Corresponding to the two types of inter-view interactions, there are two modes in our architecture: late-fusion and early-fusion memories. In the late-fusion mode, the memory space for each view is separated and independent, that is, there is no information exchange between the two memories during the encoding process. The memories' read values are only synthesised to generate inter-view knowledge in the decoding phase. Contrast to the late-fusion mode, the memory addressing space in the early-fusion mode is shared amongst views. That is, the encoder from one view can access and modify the contents of the other view's memory. This design ensures the information is shared across views via memories accessing. In order to facilitate this asynchronous sharing, we design novel cache components that temporarily hold the write values of every timestep. This enables related information at different time steps to be written to the memories together. Finally, we apply memory write-protected mechanism in the decoding process to make the inference of our model more efficient.

## 4.2 Background

### 4.2.1 Multi-View Learning

In multi-view learning, data can be naturally partitioned into channels presenting different views of the same data. Multi-view sequential learning is a sub-class of multi-view learning where each view data is in the form of sequential events, which

can be synchronous or asynchronous. In the synchronous setting, all views share the same time step and view length. Some problems of this type include video consisting of visual and audio streams; and text as a joint sequence of words and part-of-speech tags. Synchronous multi-view sequential learning is an active area (Rajagopalan et al., 2016; Zadeh et al., 2017, 2018). These works make assumptions on the time step alignment and thus they are constrained by the scope of synchronous multi-view problems.

In this work, we relax these assumptions and focus more on asynchronous settings, that is, there is no alignment amongst views and the sequence lengths vary across views. These occur when the data is collected from channels having different time scales or we cannot infer the precise time information when extracting data. In healthcare, for instance, an electronic medical record (EMR) contains information on patient's admissions, each of which consists of various views such as diagnosis, medical procedure, and medicine. Although an admission is time-stamped, medical events from each view inside the admission are not synchronous and different in length.

Asynchronous multi-view data often demonstrates three types of view interactions. The first type is intra-view interactions, those involving only one view, representing the internal dynamics. For example, each EMR view has specific rules for coding its events, forming distinctive correlations amongst medical events inside a particular view. The second type is late inter-view interactions, those that span from input views to output, representing the mapping function between the inputs and the outputs. We call it "late" because the interaction across input views is considered only in the inference process. The third type is early inter-view interactions, those that account for relations covering multiple input views and happening before the inference process. For example, in drug prescription, the diagnosis view is the cause of the medical procedure view, both of which affect the output which are medicines prescribed for patient. The interactions in sequential views not only span across views but also extend throughout the length of the sequences. One example involves patients whose diseases in current admission are related to other diseases or treatments from distant admissions in the past. The complexity of view interactions, together with the unalignment and long-term dependencies amongst views poses a great challenge in asynchronous multi-view sequential problems.

## 4.2.2 Existing Approaches

**Deep learning for healthcare:** The recent success of deep learning has drawn board interest in building AI systems to improve healthcare. Several studies have used deep learning methods to better categorise diseases and patients: denoising autoencoders, an unsupervised approach, can be used to cluster breast cancer patients (Tan et al., 2014), and convolutional neural networks (CNNs) can help count mitotic divisions, a feature that is highly correlated with disease outcome in histological images (Cireşan et al., 2013). Another branch of deep learning in healthcare is to solve biological problems such as using deep RNN to predict gene targets of microRNAs (Zurada, 1994). Despite these advances, a number of challenges exist in this area of research, most notably how to make use of other disparate types of data such as electronic medical records (EMRs). Recently, more efforts have been made to utilise EMR data in disease prediction (Pham et al., 2017), unplanned admission and risk prediction (Nguyen et al., 2017) problems. Other works apply LSTMs, both with and without attention to clinical time series for heart failure prediction (Choi et al., 2016) or diagnoses prediction (Lipton et al., 2016). Treatment recommendation is also an active research field with recent deep learning works that model EMR codes as sequence such as using sequence of billing codes for medicine suggestions (Bajor and Lasko, 2017) or using set of diagnoses for medicine sequence prediction (Zhang et al., 2017). Differing from these approaches, our works focus on modeling both the admission data and the treatment output as two sequences to capture order information from input codes and ensure dependencies amongst output codes at the same time.

**Multi-view learning in healthcare:** Multi-view learning is a well-studied problem, where methods often exploit either the consensus or the complementary principle (Xu et al., 2013). A straightforward approach is to concatenate all multiple views into one single view making it suitable for conventional machine learning algorithms, both for vector inputs (González et al., 2015; Zadeh et al., 2016) or sequential inputs (Morency et al., 2011; Song et al., 2012). Another approach is co-training (Blum and Mitchell, 1998; Nigam and Ghani, 2000), aiming to maximise the mutual agreement on views. Other approaches either establish a latent subspace shared by multiple views (Quadrianto et al., 2009) or perform multiple kernel learning (Rakotomamonjy et al., 2007). These works are typically limited to non-sequential views.

More recently, deep learning is increasingly applied for multi-view problems, especially with sequential data. For example, LSTM (Hochreiter and Schmidhuber,

1997) is extended for multi-view problems (Rajagopalan et al., 2016) or multiple kernel learning is combined with convolution networks (Poria et al., 2015). More recent methods focus on building deep networks to extract features from each view before applying different late-fusion techniques such as tensor products (Zadeh et al., 2017), contextual LSTM (Poria et al., 2017) and gated memory (Zadeh et al., 2018). All of these deep learning methods are designed only for synchronous sequential input views. Hence, the applications of these methods mostly fall into tagging problems where the output is aligned with the input views. As far as we know, the only work that can apply to asynchronous inputs is in Chung et al., (2017), in which the authors construct a dual LSTM for feature extraction and use attention for late-fusion (Chung et al., 2017). Without multiple memories for multi-view sequences, LSTM-based models fail short in capturing long-term dependencies in the sequences. LSTM encoder-decoder with attention can only support late-fusion modeling, which may be insufficient for cases that require early-fusion. More importantly, attentional LSTMs do not assume fixed size memory, which may be impractical for ultra-long sequences, online and life-long learning.

In healthcare, there are only few works that make use of multi-view data. A multi-view multi-task model is proposed to predict future chronic diseases given multi-media and multi-model observations (Nie et al., 2015). However, this model is only designed for single-instance regression problems. DeepCare (Pham et al., 2017) solves the disease progression problem by combining diagnosis and intervention views. It treats medical events in each admission as a bag and uses pooling to compute the feature vectors for the two views in an admission. The sequential property of events inside each admission is ignored and there is no mechanism to model inter-view interactions at event level. There are many other works using deep learning such as RETAIN (Choi et al., 2016), Dipole (Ma et al., 2017b) and LEAP (Zhang et al., 2017) that attack different problems in healthcare. However, they are designed for single input view.

**MANNs for healthcare:** Memory augmented neural networks (MANNs) have emerged as a new promising research topic in deep learning. Memory Networks (MemNNs) (Weston et al., 2014) and Neural Turing Machines (NTMs) (Graves et al., 2014) are the two classes of MANNs that have been applied to many problems such as meta learning (Santoro et al., 2016) and question answering (Sukhbaatar et al., 2015). In healthcare, there is limited work applying MemNN-based models to handle medical-related problems such as clinical textual QA (Hasan et al., 2016) or diagnosis inference (Prakash et al., 2017). However, these works have been using
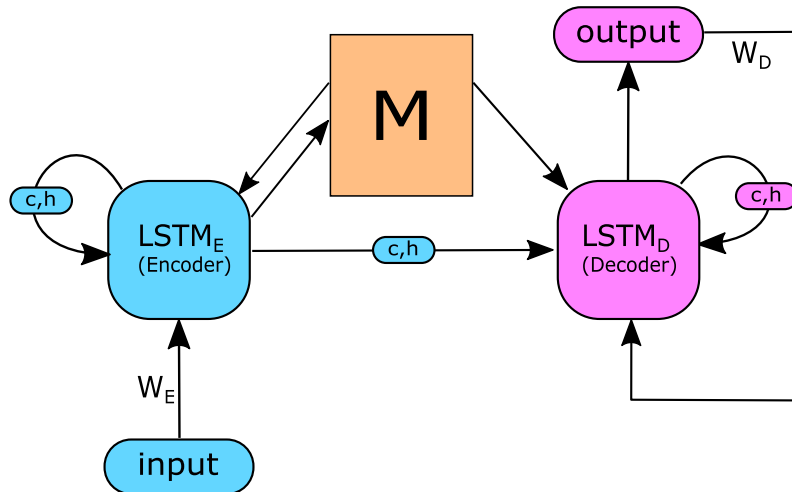
Figure 4.1: Dual Controller Write-Protected Memory Augmented Neural Network. $LSTM_E$ is the encoding controller. $LSTM_D$ is the decoding controller. Both are implemented as LSTMs.

clinical documents as input, rather than just using medical codes stored in EMRs. Our work, on the other hand, learns end-to-end from raw medical codes in EMRs by leveraging Differentiable Neural Computer (DNC) (Graves et al., 2016), the latest improvement over the NTM. In practice, DNC and other NTM variants have been used for various domains such as visual question answering (Ma et al., 2017a), and one-shot learning (Santoro et al., 2016), yet it is the first time DNC is adapted for healthcare tasks.

## 4.3   Dual Control Architecture

We now present our first contribution–a deep neural architecture called Dual Controller Write-Protected Memory Augmented Neural Network (DCw-MANN) (see Fig. 4.1). Our DCw-MANN introduces two simple but crucial modifications to the original DNC: (i) using two controllers to handle dual processes of encoding and decoding, respectively; and (ii) applying a write-protected policy in the decoding phase.

In the encoding phase, after going through embedding layer $W_E$, the input sequence is fed to the first controller (encoder) $LSTM_E$. At each time step, the controller reads from and writes to the memory information necessary for the later decoding process. In the decoding phase, the states of the first controller is passed to the second controller (decoder) $LSTM_D$. The use of two controllers instead of one is

important in our setting because it is harder for a single controller to learn many strategies at the same time. Using two controllers will make the learning easier and more focused. Also different from the encoder, the decoder can make use of its previous prediction (after embedding layer $W_D$) as the input together with the read values from the memory. Another important feature of DCw-MANN is its write-protected mechanism in the decoding phase. This has an advantage over the writing strategy used in the original DNC since at decoding step, there is no new input that is fed into the system. Of course, there remains dependencies amongst codes in the output sequence. However, as long as the dependencies amongst output codes are not too long, they can be well-captured by the cell memory $c_t$ inside the decoder's LSTM. Therefore, the decoder in our design is prohibited from writing to the memory. To be specific, at time step $t+1$ we have the hidden state and cell memory of the controllers calculated as:

$$
h_{t+1}, c_{t+1} = \begin{cases} LSTM_E\left(\left[W_E v_{d_t}, r_t\right], h_t, c_t\right); & t \leq L_{in} \\ LSTM_D\left(\left[W_D v_{p_t}, r_t\right], h_t, c_t\right); & t > L_{in} \end{cases}
\tag{4.2}
$$

where $v_{d_t}$ is the one-hot vector representing the input sequence's code at time $t \leq L_{in}$ and $v_{p_t}$ is the predicted one-hot vector output of the decoder at time $t > L_{in}$, defined as $v_{p_t} = onehot\left(o_t\right)$, i.e.,:

$$
v_{p_t}[i] = \begin{cases} 1 & ; i = \underset{1 \leqslant j \leqslant |C_p|}{argmax}(o_t[j]) \\ 0 & ; \text{otherwise} \end{cases}.
\tag{4.3}
$$

We propose a new memory update rule to enable the write-protected mechanism:

$$
M_t = \begin{cases} M_{t-1} \circ \left(E - w_t^w e_t^\top\right) + w_t^w v_t^\top; & t \leq L_{in} \\ M_{t-1}; & t > L_{in} \end{cases}
\tag{4.4}
$$

where $E$ is an $N \times D$ matrix of ones , $w_t^w \in [0,1]^N$ is the write-weight, $e_t \in [0,1]^D$ is an erase vector, $v_t \in \mathbb{R}^D$ is a write vector, $\circ$ is point-wise multiplication, and $L_{in}$ is the length of input sequence.
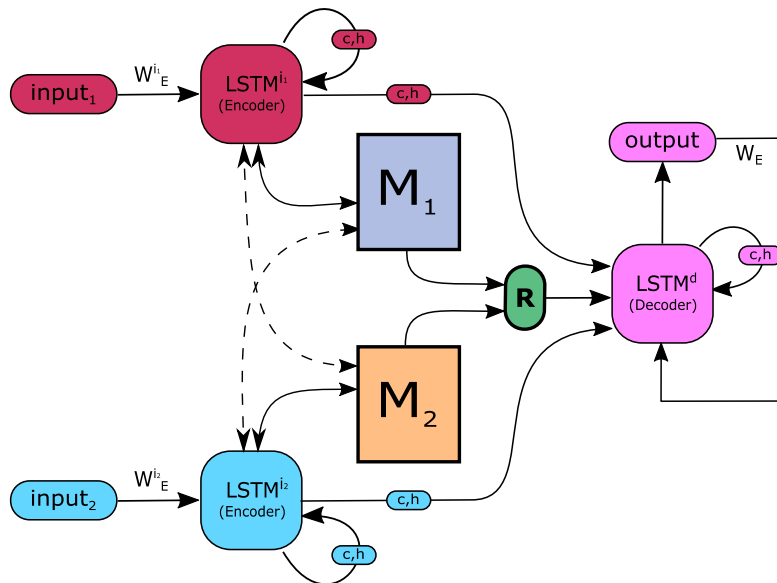
Figure 4.2: Dual Memory Neural Computer. $LSTM^{i_1}$, $LSTM^{i_2}$ are the two encoding controllers implemented as LSTMs. $LSTM^d$ is the decoding controller. The dash arrows represent cross-memory accessing in early-fusion mode.

## 4.4 Dual Memory Architecture

We now present the second contribution to solve the generic asynchronous two-view sequential learning: a new deep memory augmented neural network called Dual Memory Neural Computer (DMNC).

### 4.4.1 Dual Memory Neural Computer

Our architecture consists of three neural controllers (two for encoding and one for decoding), each of which interacts with two external memory modules (see Fig. 4.2). Each of the two memory modules is similar to the external memory module in DNC (Graves et al., 2016), that is, it is equipped with temporal linkage and dynamic allocation. The three controllers have their own embedding matrices $W_E^{i_1}$, $W_E^{i_2}$, $W_E$ which project the one-hot representation of events to a unified $d$-dimensional space. We use $\mathbf{x}_{t_1}^{i_1}$, $\mathbf{x}_{t_2}^{i_2}$, $\mathbf{y}_t \in \mathbb{R}^d$ to denote the embedding vector of $x_{t_1}^{i_1}$, $x_{t_2}^{i_2}$, $y_t$, respectively, in which $\mathbf{x}_{t_1}^{i_1} = W_E^{i_1} x_{t_1}^{i_1}, \mathbf{x}_{t_2}^{i_2} = W_E^{i_2} x_{t_2}^{i_2}, \mathbf{y}_t = W_E y_t$. The embedding vectors $\mathbf{x}_{t_1}^{i_1}$, $\mathbf{x}_{t_2}^{i_2}$ are always used as inputs of the encoders while the embedding vector $\mathbf{y}_t$ will only be used as input of the decoder if the output view is a sequence.

Each encoder will transform the embedding vectors to $h$-dimensional hidden vectors. The current hidden vectors and outputs of the encoders are computed as:

$$h_{t_1}^{i_1}, o_{t_1}^{i_1} = LSTM^{i_1}\left(\left[\mathbf{x}_{t_1}^{i_1}, r_{t_1-1}^{i_1}\right], h_{t_1-1}^{i_1}\right), 1 \le t_1 < L^{i_1} \tag{4.5}$$

$$h_{t_2}^{i_2}, o_{t_2}^{i_2} = LSTM^{i_2}\left(\left[\mathbf{x}_{t_2}^{i_2}, r_{t_2-1}^{i_2}\right], h_{t_2-1}^{i_2}\right), 1 \le t_2 < L^{i_2} \tag{4.6}$$

where $r_{t_1-1}^{i_1}$, $r_{t_2-1}^{i_2}$ are read vectors at previous time step of each encoder and $L^{i_1}, L^{i_2}$ are the lengths of input views. It should be noted that the time step in each view may be asynchronous and the lengths may be different. In our applications, since we treat input views as sequences, we use $LSTM$ as the core of the encoders[2]. Using separated encoder for each view naturally encourages the intra-view interactions. To model inter-view interactions, we use two modes of memories, late-fusion and early-fusion.

**Late-fusion memories:** In this mode, our architecture only models late inter-view interactions. In particular, $r_{t_1}^{i_1}$ and $r_{t_2}^{i_2}$ are computed separately:

$$r_{t_1}^{i_1} = \left[r_{t_1}^{i_{1,1}}, ..., r_{t_1}^{i_{1,R}}\right] = m_{read}^{e_1}\left(o_{t_1}^{i_1}, M_1\right) \tag{4.7}$$

$$r_{t_2}^{i_2} = \left[r_{t_2}^{i_{2,1}}, ..., r_{t_2}^{i_{2,R}}\right] = m_{read}^{e_2}\left(o_{t_2}^{i_2}, M_2\right) \tag{4.8}$$

where $M_1$, $M_2$ are the two memory matrices containing view-specific contents and $m_{read}^{e_1}$, $m_{read}^{e_2}$ are two read functions of the encoders with separated set of parameters. Given the encoder output vectors, the read functions produce the keys $k_{t_1}^{i_1}$, $k_{t_2}^{i_2}$ in the manner of DNC. The keys are used to address the corresponding memory and compute the read vectors using Eq.(3.40). This design ensures the dynamics of computation in one view does not affect the other's and only in-view contents are stored in view-specific memory. This mode is important because in certain situations, writing external contents to view-specific memory will interfere the acquired knowledge and obstruct the learning process. In Section 4.1, we will show a case study that fits with this setting and the empirical results will demonstrate that the late-fusion mode is necessary to achieve better performance.

---

[2]For inputs as sets, we can replace the $LSTM$s with $MLP$s

**Early-fusion memories:** When there exists a strong correlation between the two input views, requiring to model early inter-view interactions, we introduce another mode of memories: early-fusion mode. In this mode, the two memories share the same addressing space, that is, the encoder from one view can access the memory content from another view and vice versa. Also, the read functions $m_{read}^e$ share the same parameter set:

$$r_{t_1}^{i_1} = \left[r_{t_1}^{i_1,1}, ..., r_{t_1}^{i_1,R}\right] = m_{read}^e\left(o_{t_1}^{i_1}, [M_1, M_2]\right) \tag{4.9}$$

$$r_{t_2}^{i_2} = \left[r_{t_2}^{i_2,1}, ..., r_{t_2}^{i_2,R}\right] = m_{read}^e\left(o_{t_2}^{i_2}, [M_1, M_2]\right) \tag{4.10}$$

Since the read vectors for one encoder can come from either memories, the encoder's next hidden values are dependent on both views' memory contents, which enables possible early inter-view interactions in this mode.

**Memories modification with cache components:** In both modes, the two memories are updated every timestep by the two encoders. While in the late-fusion mode, the writings to two memories are independent and can be executed in parallel using Eq.(3.36), in the early-fusion mode, the writings must be executed in an alternating manner. In particular, the two encoders take turn writing to memories, allowing the exchange of information at every timestep. Doing this way is optimal if the two views are synchronous and equal in lengths. To make it work with variable length input views, we introduce a new component to our architecture: a cache memory that lies between the controller and the external memory. Different from the original DNC which writes directly the event's value to the external memory, in the early-fusion mode of our architecture, each controller integrates write values inside its own cache memory $c_t$ until an appropriate moment before committing them to the external memory. We introduce $g_t^c$ as a learnable cache gate to control the degree of integration between current write value and the previous cache's content as follows:

$$g_t^c = f^c\left(o_t^i\right) \tag{4.11}$$

$$c_t = g_t^c \circ c_{t-1} + (1 - g_t^c) \circ v_t \tag{4.12}$$

In these equations, $g_t^c$ is the cache gate, $o_t^i$ is the encoder output, $f^c$ is a learnable

---

**Algorithm 4.1** Training algorithm for healthcare data (set output)

---

**Require:** Training set $\{\{(X_a^{i_1}, X_a^{i_2}, Y_a)\}_{a=1}^A\}_{n=1}^N$

1:  Sample $B$ samples from training set
2:  **for each** sample in $B$ **do**
3:      Clear memory $M_1, M_2$
4:      **for** $a = 1, A$ **do**
5:          $(X^{i_1}, X^{i_2}, Y) = (X_a^{i_1}, X_a^{i_2}, Y_a)$
6:          **while** $t_1 < L^{i_1}$ or $t_2 < L^{i_2}$ **do**
7:              **if** $t_1 < L^{i_1}$ **then**
8:                  Use Eq.(4.5) to calculate $h_{t_1}^{i_1}, o_{t_1}^{i_1}$
9:                  Use Eq.(3.36) or Eq.(4.13) to update $M_1$
10:                 Use Eq.(4.7) or Eq.(4.9) to read $M_1$
11:                 $t_1 = t_1 + 1$
12:             **end if**
13:             **if** $t_2 < L^{i_2}$ **then**
14:                 Use Eq.(4.6) to calculate $h_{t_2}^{i_2}, o_{t_2}^{i_2}$
15:                 Use Eq.(3.36) or Eq.(4.13) to update $M_2$
16:                 Use Eq.(4.8) or Eq.(4.10) to read $M_2$
17:                 $t_2 = t_2 + 1$
18:             **end if**
19:         **end while**
20:         Use Eq.(4.14) and Eq.(4.15) to read $M_1, M_2$
21:         Use Eq.(4.18) to calculate $\widehat{y}$
22:         Update parameter $\theta$ using $\nabla_\theta Loss_{set}(Y, \widehat{y})$
23:     **end for**
24: **end for**

---

function[3], $c_t$ is the cache content and $v_t$ is the write value. Then, the cache will be written to the memory using the following formula:

$$M_t = M_{t-1} \circ \left(E - g_t^w w_t^w e_t^\top\right) + g_t^w w_t^w c_t^\top \tag{4.13}$$

We propose this new writing mechanism for early-fusion mode to enable one encoder to wait for another while processing input events (in this context, waiting means the encoder stops writing to memory). In the original DNC, if the write gate $g_t^w$ is close to zero, the encoder does not write to memory and the write value at current time step will be lost. However, in our design, even when there is no writing, the write value somehow can be kept in the cache if $g_t^c < 1$. The cache in a view may choose to hold an event's write value instead of writing it immediately at the read time step. Thus, the information of the event is compressed in the cache until appropriate

---

[3]In this section, all $f$ functions are implemented as single-layer feed-forward neural networks

occasion, which may be after the appearance of another event from the other view. This mechanism enables two related asynchronous events to simultaneously involve in building up the memories.

**Write-protected memories:** In our architecture, during the inference process, the decoder stops writing to memories. We add this feature to our design because the decoder does not receive any new input when producing output. Writing to memories in this phase may deteriorate the memory contents, hampering the efficiency of the model.

### 4.4.2 Inference in DMNC

In this section, we give more details on the operation of the decoder. Because the decoder works differently for different output types (set or sequence), we will present two versions of decoder implementation.

**Output as sequence:** In this setting, the decoder ingests the encoders' final states as its initial hidden state $h_0 = \left[ h_{L^{i_1}}^{i_1}, h_{L^{i_2}}^{i_2} \right]$. The decoder's hidden and output vectors are given as: $h_t, [o_t^1, o_t^2] = LSTM^d \left( \left[ \mathbf{y}_{t-1}^*, r_{t-1}^{i_1}, r_{t-1}^{i_2} \right], h_{t-1} \right)$. Here, $\mathbf{y}_{t-1}^*$ is the embedding of the previous prediction $y_{t-1}^*$. The decoder combines the read vectors from both memories to produce a probability distribution over the output:

$$r_t^{i_1} = \left[ r_t^{i_1,1}, ..., r_t^{i_1,R} \right] = m_{read}^d \left( o_t^1, M_1 \right) \tag{4.14}$$

$$r_t^{i_2} = \left[ r_t^{i_2,1}, ..., r_t^{i_2,R} \right] = m_{read}^d \left( o_t^2, M_2 \right) \tag{4.15}$$

$$P\left( y_t | X^{i_1}, X^{i_2} \right) = \pi \left( \left[ o_t^1, o_t^2 \right] + f^d \left( \left[ r_t^{i_1}, r_t^{i_2} \right] \right) \right) \tag{4.16}$$

where $r_t^{i_1}, r_t^{i_2}$ are read vector from $M_1, M_2$, respectively, provided by the read function $m_{read}^d$, $f^d$ is a learnable function and $\pi$ is softmax function. The current prediction is $y_t^* = \underset{y \in S}{argmax}\, P\left( y_t = y | X^{i_1}, X^{i_2} \right)$ and the loss function is the cross entropy:

$$Loss_{seq}\left( Y, P \right) = - \sum_{t=1}^{L} \log P\left( y_t | X^{i_1}, X^{i_2} \right) \tag{4.17}$$

**Output as set:** In this setting, the decoder uses $m_{read}^d$ to read from the memories once to get the read vectors $r^{i_1}, r^{i_2}$. The decoder combines these vectors with the encoders' final hidden values to produce the output vector $\hat{y} \in \mathbb{R}^{|S|}$:

$$\hat{y} = \sigma \left( f^d (W_1 r^{i_1} + W_2 r^{i_2} + W_3 \left[ h_{L^{i_1}}^{i_1}, h_{L^{i_2}}^{i_2} \right]) \right) \qquad (4.18)$$

Here, the combination is simply the linear weighted summation with parameter matrices $W_1$, $W_2$, $W_3$. $f^d$ is a learnable function and $\sigma$ is the sigmoid function. For set output, the loss function is multi-label loss defined as:

$$Loss_{set}(Y, \hat{y}) = - \left( \sum_{y_l \in Y} \log \hat{y}_l + \sum_{y_l \notin Y} \log (1 - \hat{y}_l) \right) \qquad (4.19)$$

For both settings, the decoder makes use of both memories' contents and encoders' final hidden values to produce the output. While memory contents represent the long-term knowledge, the encoder's hidden values represent the short-term information stored inside the controllers. Both are crucial to model inter-view interactions and necessary for the decoder to predict the correct outputs.

### 4.4.3 Persistent Memory for Multiple Admissions

As mentioned earlier in Sec. 4.1.2, one unique property of healthcare is the long-term dependencies amongst admissions. Therefore, the output at the current admission $Y_a$ is dependent on the current and all previous admission's inputs $\left\{ \left( X_{pa}^{i_1}, X_{pa}^{i_1} \right) \right\}_{pa=1}^{a}$. There are several ways to model this property. The simplest solution is to concatenate the current admission with previous ones to make up single sequence input for the model. This method causes data replication and preprocessing overhead. Another solution is to use recurrent neural network to model the dependencies. Some papers use GRU and LSTM where each time step is fed with an admission. The admission is treated as a set of medical events and represented by a feature vector (Choi et al., 2015; Pham et al., 2017).

In our memory-augmented architecture, we can model this dependencies by using the memories to store information from previous admissions. In the original DNC, the memory content is flushed every time new data sample (i.e. new admission) is fed – this certainly loses the information of admission history. We modify this mechanism by keeping the memories persistent during a patient's admissions processing. That

is, the content of memories is built up and modified during the whole history of a patient's admissions. The memories are only cleared prior to reading a new patient's record.

Persistent memories in our architecture play two important roles. First, because the number of events across admissions are large while memory sizes are moderate, the memory modules learn to compress efficiently the input views, keeping only essential information. This makes memory look-ups in the decoding process only limited to a fixed size of chosen knowledge. This is more compact and focused than attention mechanisms, in which the decoder has to attend to all events in the input. Second, each memory slot can store information of any event in the input views, which enables skip-connection reference in the decoding process, i.e., the decoder can jump to any input event, even the one in the farthest admission, to look for relevant information. The whole process of training our dual memory neural computer for healthcare data is summarised in Algorithm 4.1.

## 4.5  Applications

In this section, we perform experiments both on real-world data and synthetic tasks. The purpose of the synthetic task is to study the incremental impact of dual control modifications we propose. Moreover, we demonstrate the effectiveness of our proposed dual memory model DMNC. We use $DMNC_l$ and $DMNC_e$ to denote the late-fusion and early-fusion mode of our model, respectively. The data for real-world problems are real EMR data sets, some are public accessible. We make the source code of DCw-MANN and DMNC publicly available at https://github.com/thaihungle/MAED and https://github.com/thaihungle/DMNC, respectively.

### 4.5.1  Synthetic Task: Odd-Even Sequence Prediction

In this task, the input is sequence of random odd numbers chosen without replacement from the set $S_o = \{1, 3, 5, ..., 49\}$ and the output is sequence of even numbers from the set $S_e = \{2, 4, 6, ..98\}$. The $n$-th number $y_n$ in the output sequence is computed as:
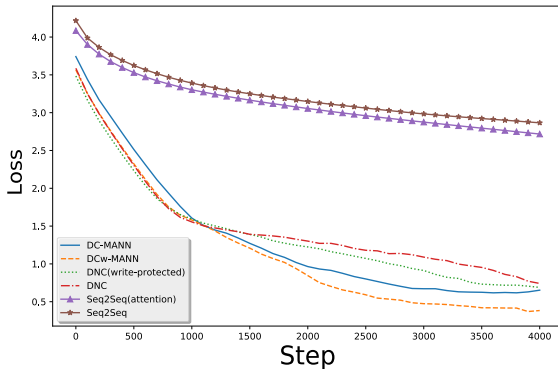
Figure 4.3: Training Loss of Odd-Even Task
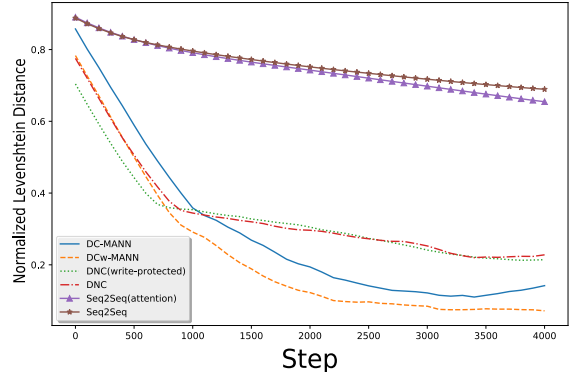
Figure 4.4: Training NLD of Odd-Even Task

$$y_n = \begin{cases} 2x_n & n \leq \left\lfloor \frac{L}{2} \right\rfloor \\ y_{n-1} + 2 & n > \left\lfloor \frac{L}{2} \right\rfloor \end{cases}$$ . $x_n$ is the $n$-th number in the input sequence and $L$ is the length of both input and output sequence chosen randomly from the range $[1, 20]$. The formula is designed to reflect healthcare situations where treatment options depend both on diagnoses in the input sequence and other treatments in the same output sequence. Here is an example of an input-output sequence pair with $L = 7$: $input := [11, 7, 25, 39, 31, 1, 13]$ and $output := [22, 14, 50, 52, 54, 56, 58]$. We want to predict the even numbers in the output sequence given odd numbers in the input sequence, hence we name it odd-even prediction task. In this task, the model has to "remember" the first half of the input sequence to compute the first half of the output sequence, then it should switch from using input to using previous output at the middle of the output sequence to predict the second half.

**Evaluations:** Our baselines are Seq2Seq (Sutskever et al., 2014a), its attention version (Bahdanau et al., 2015) and the original DNC (Graves et al., 2016). Since we want to analyse the impact of new modifications, in this task, we explore two other models: DNC with write-protected mechanism in the decoding phase and dual controller MANN without write-protected mechanism (DC-MANN). We use the Levenshtein distance (edit distance) to measure the model's performance. To account for variable sequence lengths, we normalise this distance over the length of the longer sequence (between 2 sequences). The predicted sequence is good if its Normalised Levenshtein Distance (NLD) to the target sequence is small.

**Implementation details:** For all experiments, deep learning models are implemented in Tensorflow 1.3.0. Optimiser is Adam (Kingma and Ba, 2014) with learning rate of 0.001 and other default parameters. The hidden dimensions for LSTM

| Model | NLD |
|---|---|
| Seq2Seq | 0.679 |
| Seq2Seq with attention | 0.637 |
| DNC | 0.267 |
| DNC (write-protected) | 0.250 |
| DC-MANN | 0.161 |
| DCw-MANN | **0.082** |

Table 4.1: Test Results on Odd-Even Task (lower is better)
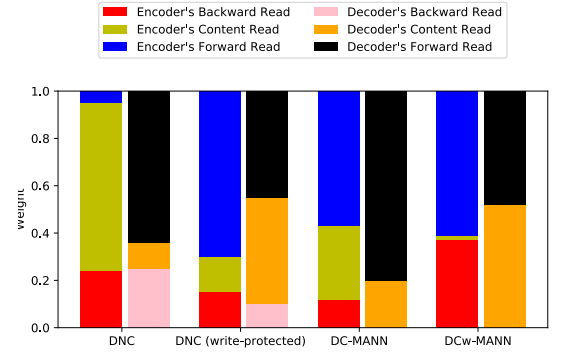


Figure 4.5: Read Modes of MANNs on Odd-Even Task

and the embedding sizes for all models are set to 256 and 64, respectively. Memory's parameters including number of memory slots and the size of each slot are set to 128 and 128 , respectively.

**Results:** After training with 4000 input-output pair of sequences, the models will be tested for the next 1000 pairs. The learning curves of the models are plotted in Figs. 4.3 and 4.4. The average NLD of the predictions is summarised in Table 4.1. As is clearly shown, the proposed model outperforms other methods. Seq2Seq-based methods fail to capture the data pattern and underperform other methods. The introduction of two controllers helps boost the performance of DNC significantly. Additional DNC-variant with write-protected also performs better than the original one, which suggests the benefit of decoding without writing.

Fig. 4.5 plots read mode weights for three reading strategies employed in encoding and decoding phases. We can observe the differences in the way the models prefer reading strategies. The biggest failure of DNC is to keep using backward read in the decoding process. This is redundant because in this problem, it is the forward of the previous read location (if the memory location that corresponds to $x_{n-1}$ is the previous read, then its forward is the memory location that corresponds to $x_n$) that defines the current output ($y_n$). On the other hand, dual controllers with write-protected mechanism seems help the model avoid bad strategies and focus more on learning reasonable strategies. For example, using dual controllers tends to lessen the usage of content-based read in the encoding phase. This strategy is reasonable in this example since the input at each time step is not repeated. Write-protected policy helps balance the forward and content-based read in the decoding phase, which may reflect the output pattern – half-dependent on the input and half-dependent on the

| MIMIC-III Dataset (# of visit >1) | Procedure as output | Drug as output |
| --- | --- | --- |
| # of patients | 6,314 | 5,620 |
| # of admissions | 16,317 | 14,656 |
| # of unique diagnosis codes | 4,669 | 4,563 |
| # of unique treatment codes | 1,439 | 2,446 |
| Average # of diagnosis sequence length | 13.3 | 13.8 |
| Max # of diagnosis sequence length | 39 | 39 |
| Average # of treatment sequence length | 4.7 | 11.4 |
| Max # of treatment sequence length | 40 | 186 |
| Average # of visits per patient | 2.5 | 2.6 |
| Max # of visits per patient | 29 | 29 |

Table 4.2: Statistics of MIMIC-III sub-datasets

previous output.

## 4.5.2   Treatment Recommendation Tasks

The dataset used for this task is MIMIC-III (Johnson et al., 2016), which is a publicly available dataset consisting of more than 58k EMR admissions from more than 46k patients. An admission history in this dataset can contain hundreds of medical codes, which raises a great challenge in handling long-term dependencies. In MIMIC-III, there are both procedure and drug codes for the treatment process so we consider two separate treatment recommendation tasks: procedure prediction and drug prescription. In practice, if we use all the drug codes in an EMR record, the drug sequence can be very long since, each day in hospital, the doctor can prescribe several types of drugs for the patient. Hence, we only pick the first drug used in a day during the admission as the representative drug for that day. We also follow the previous practice that only focuses on patients who have more than one visit (Ma et al., 2017b; Nguyen et al., 2017; Pham et al., 2017). The statistics of the two sub-datasets is detailed in Table 4.2.

**Evaluations:** For comprehensiveness, beside direct competitors, we also compare our methods with classical for healthcare predictions, which are Logistic Regression and Random Forests. Because traditional methods are not designed for sequence predictions, we simply pick the top outputs (ignoring ordering information). In treatment recommendation tasks, we use precision, which is defined as the number of correct predicted treatment codes (ignoring the order) divided by the number of predict treatment codes. More formally, let $S_p^n$ be the set of ground truth treatments

| Model | Procedure Output | | Drug Output | |
|---|---|---|---|---|
| | Precision | Jaccard | Precision | Jaccard |
| Logistic Regression | 0.256 | 0.185 | 0.412 | 0.311 |
| Random Forest | 0.276 | 0.199 | 0.491 | 0.405 |
| Seq2Seq | 0.263 | 0.196 | 0.220 | 0.138 |
| Seq2Seq with attention | 0.272 | 0.204 | 0.224 | 0.142 |
| DNC | 0.285 | 0.214 | 0.577 | 0.529 |
| DCw-MANN | **0.292** | **0.221** | **0.598** | **0.556** |

Table 4.3: Results on MIMIC-III dataset for procedure prediction and drug prescription (higher is better).

for the $n$-th admission, $S_q^n$ be the set of treatments that the model outputs. Then the precision is: $\frac{1}{N} \sum_{n=1}^{N} \frac{|S_p^n \cap S_q^n|}{|S_q^n|}$, where $N$ is total number of test patients. To measure how closely the generated treatment compares against the real treatment, we use Mean Jaccard Coefficient[4], which is defined as the size of the intersection divided by the size of the union of ground truth treatment set and predicted treatment set: $\frac{1}{N} \sum_{n=1}^{N} \frac{|S_p^n \cap S_q^n|}{|S_p^n \cup S_q^n|}$.

**Implementation details:** We randomly divide the dataset into the training, validation and testing set in a 0.7 : 0.1 : 0.2 ratio, where the validation set is used to tune model's hyper-parameters. For the classical Random Forests and Logistic Classifier, the input is bag-of-words. Also, we apply One-vs-Rest strategy (Rifkin and Klautau, 2004) to enable these classifiers to handle multi-label output and the hyper-parameters are found by grid-searching.

**Results:** Table 4.3 reports the prediction results on two tasks (procedure prediction and drug prescription). The performance of the proposed DCw-MANN is higher than that of baselines on the testing data for both tasks, validating the use of dual controllers with write-protected mechanism. Without memory, Seq2Seq methods seem unable to outperform classical methods, possibly because the evaluations are set-based, not sequence-based. In the drug prescription task, there is a huge drop in performance of the Seq2Seq-based approaches. It should be noted that, in drug prescription, the drug codes are given day by day; hence, the average length of output sequence are much longer than the procedure's one. This could be a very challenging task for Seq2Seq. Memory-augmented models, on the other hand, have an external memory to store information, so it can cope with long-term dependencies. Figs. 4.6 and Fig. 4.7 show that compared to DNC, DCw-MANN is the faster learner in drug

---

[4]The metrics actually are at disadvantage to the proposed sequence-to-sequence model, but we use to make them easy to compare against non-sequential methods.
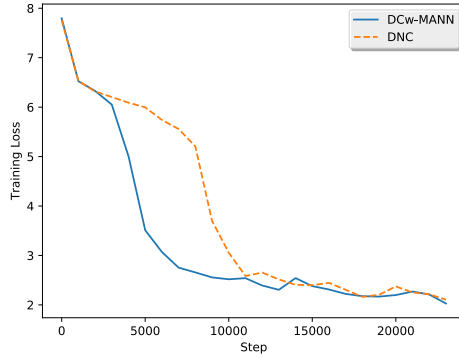
Figure 4.6: Training Loss of Drug Prescription Task
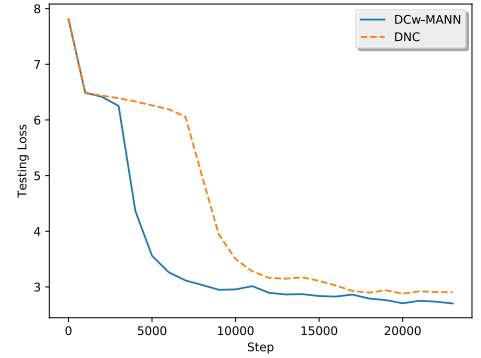


Figure 4.7: Testing Loss of Drug Prescription Task

Table 4.4: Sum of two sequences task test results. Max train sequence length is 10.

| Model | Accuracy (%) | | |
|---|---|---|---|
| | $L_{max} = 10$ | $L_{max} = 15$ | $L_{max} = 20$ |
| LSTM | 35.17 | 24.12 | 18.64 |
| DNC | 37.8 | 20.43 | 14.67 |
| Dual LSTM | 52.41 | 42.57 | 30.47 |
| WLAS | 55.98 | 43.29 | 32.49 |
| $DMNC_l$ | **99.76** | **98.53** | **78.17** |
| $DMNC_e$ | 98.84 | 93.00 | 69.93 |

prescription task. This case study demonstrates that a MANN with dual controller and write-protected mechanism can significantly improve the performance of the sequence prediction task in healthcare.

## 4.5.3  Synthetic Task: Sum of Two Sequences

We conduct this synthetic experiment to verify our model performance and behavior. In this problem, the input views are two randomly generated sequence of numbers: $\{x_1^1, ..., x_L^1\}$, $\{x_1^2, ..., x_L^2\}$. Each sequence has $L$ integer numbers. $L$ is randomly chosen from range $[1, L_{max}]$ and the numbers are randomly chosen from range $[1, 50]$. The output view is also a sequence of integer numbers defined as $\left\{y_i = x_i^1 + x_{L+1-i}^2\right\}_{i=1}^{L}$, in which $y_i \in [2, 100]$. Note that this summation form is unknown to the model. During training, only the outputs are given. Because the output's number is the sum of two numbers from the two input views, we name the
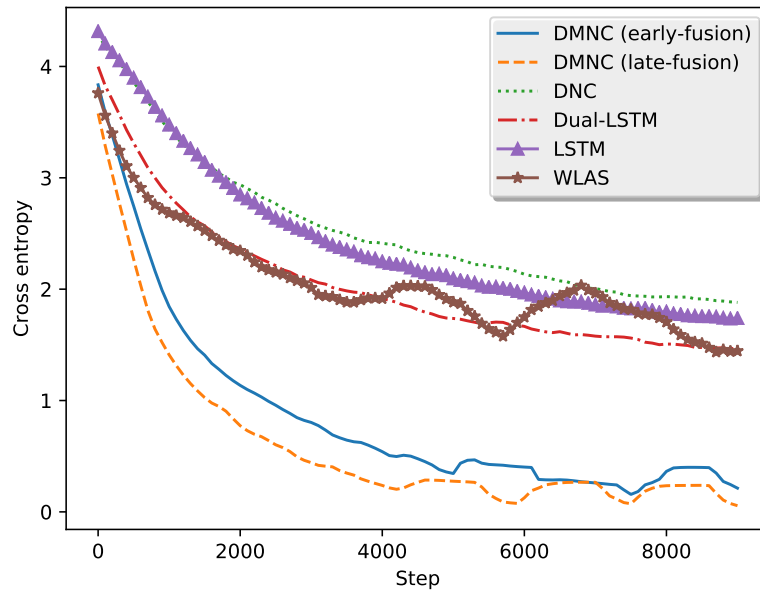
Figure 4.8: Training loss of sum of two sequences task. The training error curves have similar patterns.

task as sum of two sequences. It should be noted that two input numbers in the summation do not share the same time step; hence, the problem is asynchronous. To learn and solve the task, a model has to read all the numbers from the two input sequences and discover the correct pair that will be used to produce the summation. Synchronous multi-view models certainly fail this task because they assume the inputs to be aligned. In the training phase, we choose $L_{max} = 10$, training for 10,000 iterations with mini batch size $= 50$. In the testing phase, we evaluate on 2500 random samples with $L_{max} = 10$, $L_{max} = 15$, $L_{max} = 20$ to verify the generalisation of the models beyond the range where they are trained.

**Evaluations**: the baselines for this synthetic task are chosen as follows:

- View-concatenated sequential models: This concatenates events in input views to form one long sequence. This technique transforms the two-view sequential problem to normal sequence-to-sequence problem. We pick LSTM and DNC as two representative methods for this approach.

- Attention model WLAS (Chung et al., 2017): This has a LSTM encoder per view, and attention is used for decoding, similar to that in machine translation (Cho et al., 2014a; Bahdanau et al., 2015). The model is applied successfully in the problem of video sentiment analysis. To make it suitable for our tasks,

we replace the encoders' feature-extraction layers in the original WLAS by an embedding layer. We choose this model as baseline since its architecture is somehow similar to ours. The difference is that we make use of external memories instead of attention mechanism.

- Dual LSTM: This model is the WLAS model without attention, that is, only the final states of encoders are passed into the decoder.

**Implementations:** For all models, embedding and hidden dimensions are 64 and 128, respectively. Word size for memory-based methods are 64. Memory size for the view-concatenated DNC and DMNC are 32 and 16, respectively. We double the memory size for view-concatenated DNC to account for the fact that the length of the input sequence is nearly double due to view concatenation. We use Adam optimiser with default parameters and apply gradient clipping size = 10 to train all models. Since output is a sequence, we use the cross-entropy loss function in Eq.(4.17). The evaluation metric used in this task is accuracy – the number of correct predictions over the length of output sequence.

**Results:** The training loss curves of the models are plotted in Fig. 4.8. The test average accuracy is summarised in Table 4.4. As clearly shown, overall the proposed model outperforms other methods by a huge margin of about 45%. Although dual LSTM and WLAS perform better than view-concatenated methods, it's too hard for non-memory methods to "remember" correctly pairs of inputs for later output summation. View-concatenated DNC even with double memory size still fails to learn the sum rule because storing two views' data in a single memory seems to mess up the information, making this model perform worst. Between two versions of DMNC, late-fusion mode is better perhaps due to the independence between two inputs' number sequences. This is the occasion where trying to model early cross-interactions damages the performance. The slight drop in performance when testing with $L_{max} = 15$ shows that our model really learns the sum rule. When $L_{max} = 20$, the input length is longer than the memory size, so even when DMNCs can learn the sum rule, they cannot store all input pairs for later summation. However, our methods still manage to perform better than any other baseline.

Table 4.5: MIMIC-III data statistics.

| # of admissions | 42,586 | # of diag | 6,461 |
|---|---|---|---|
| # of patients | 34,594 | # of proc | 1,881 |
| Avg. view len | 53.86 | # of drug | 300 |

Table 4.6: Mimic-III drug prescription test results.

| Model | AUC | F1 | P@1 | P@2 | P@5 |
|---|---|---|---|---|---|
| | | Diagnosis Only | | | |
| Binary Relevance | 82.6 | 69.1 | 79.9 | 77.1 | 70.3 |
| Classifier Chains | 66.8 | 63.8 | 68.3 | 66.8 | 61.1 |
| LSTM | 84.9 | 70.9 | 90.8 | 86.7 | 79.1 |
| DNC | 85.4 | 71.4 | 90.0 | 86.7 | 79.8 |
| | | Procedure Only | | | |
| Binary Relevance | 81.8 | 69.4 | 82.6 | 80.1 | 73.6 |
| Classifier Chains | 63.4 | 61.7 | 83.7 | 80.3 | 71.9 |
| LSTM | 83.9 | 70.8 | 88.1 | 86.0 | 78.4 |
| DNC | 83.2 | 70.4 | 88.4 | 85.8 | 78.7 |
| | | Diagnosis and procedure | | | |
| Binary Relevance | 84.1 | 70.3 | 81.0 | 78.2 | 72.3 |
| Classifier Chains | 64.6 | 63.0 | 84.6 | 81.5 | 74.2 |
| LSTM | 85.8 | 72.1 | 91.6 | 86.8 | 80.5 |
| DNC | 86.4 | 72.4 | 90.9 | 87.4 | 80.6 |
| Dual LSTM | 85.4 | 71.4 | 90.6 | 87.1 | 80.5 |
| WLAS | 86.6 | 72.5 | 91.9 | 88.1 | 80.9 |
| $DMNC_l$ | 87.4 | 73.2 | **92.4** | 88.9 | **82.6** |
| $DMNC_e$ | **87.6** | **73.4** | 92.1 | **89.9** | 82.5 |

## 4.5.4   Drug Prescription Task

The data set used for this task is MIMIC-III, which is a publicly available dataset consisting of more than 52k EMR admissions from more than 46k patients. In this task, we keep all the diagnosis and procedure codes and only preprocess the drug code since the raw drug view's average length can reach hundreds of codes in an admission, which is too long given the amount of data. Therefore, only top 300 frequently used of total 4781 drug types are kept (covering more than 70% of the raw data). The final statistics of the preprocessed data is summarised in Table 4.5.

**Evaluations:** We compare our model with the following baselines:

- Bag of words and traditional classifiers: In this approach, each input view is considered as a set of events. The vector represents the view is the sum of one-hot vectors representing the events. These view vectors are then concatenated and passed into traditional classifiers: SVM, Logistic Regression, Random Forest. To help traditional methods handle multi-label output, we apply two popular techniques: Binary Relevance (Luaces et al., 2012) and Classifier Chains (Read et al., 2011). We will only report the best model for each of the two techniques, which are Logistic Regression and Random Forest, respectively.

- View-concatenated sequential models (LSTM, DNC), Dual LSTM and WLAS (Chung et al., 2017): similar to those described in the synthetic task.

- Single-view models: To see the performance gains when making use of two input views, we also report results when only using one view for Binary Relevance, Classifier Chains, LSTM and DNC.

**Implementations:** We randomly divide the dataset into the training, validation and testing set in a $2/3 : 1/6 : 1/6$ ratio. For traditional methods, we use grid-searching over typical ranges of hyper-parameters to search for best hyper-parameter values. Deep learning models' best embedding and hidden dimensions are 64 and 64, respectively. Optimal word and memory size for DMNC are 64 and 16, respectively. The view-concatenated DNC shares the same setting except the memory size is doubled to 32 memory slots. Since the output in this task is a set, we use the multi-label loss function in Eq.(4.19) for deep learning methods. We measure the relative quality of model performances by using common multi-label metrics, Area Under the ROC Curve (AUC) and F1 scores, both of which are macro-averaged. Similar results can be achieved when using micro-averaged so we did not report them here. In practice, precision at $k$ (P@$k$) are often used to judge the treatment recommendation quality. Therefore, we also include them ($k = 1, 2, 5$) in the evaluation metrics.

**Results:** Table 4.6 shows the performance of experimental models on aforementioned performance metrics. We can see the benefit of using two input views instead of one, which helps improve the model performances. Traditional methods clearly underperform deep learning methods perhaps because these methods are hard to scale when there are many output labels and the inputs in our problem are not bag-of-words. Amongst deep learning models, our proposed ones consistently outperform others in all type of measurements. Our methods demonstrate 1-2% improvements over the second runner-up baseline WLAS. The late-fusion mode seems suitable for

certain type of metrics, but overall, the early-fusion mode is the winner, highlighting the importance of modeling early inter-view interactions.

**Case study:** In Table 4.7, we show an example of drugs prescribed for a patient given his current diagnoses and procedures. The patient had serious problems with his bowel as described in the first four diagnoses. The next three diagnoses are also severe relating to his heart problems while the remaining diagnoses are less urgent. It seems that heart-related diagnoses later led to heart surgeries listed in the procedure codes. Both modes of DMNC predict correctly the drug Docusate Sodium used to cure urgent bowel symptoms. Relating to heart diseases and surgeries, our models predict closely to expert's choices. Potassium Chloride is necessary for a healthy heart. Acetaminophen and Propofol are commonly used during surgeries. However, some heart medicine such as Heparin is missed by the two models. Figs. 4.9 and 4.10 demonstrate the "focus" of the two memories on diagnosis and procedure view, respectively. The higher the write gate values, the more information of the medical codes will be written into the memories. We can see both modes pay less attention on last diagnoses corresponding to less severe symptoms. Compared to the late-fusion, the early-fusion mode keeps more information on procedures, especially the heart-related events. This may help increase the weight on heart-related medicines and enable it to include Acetylsalicylic Acid, a common drug used after heart attack in the top recommendations.

### 4.5.5   Disease Progression Task

Data used in this task are two chronic cohorts of diabetes and mental EMRs collected between 2002-2013 from a large regional hospital in Australia. Since we want to predict the next diagnoses for a patient given his or her history of admission, we preprocessed the datasets by removing patients with less than 2 admissions, which ends up with 53,208 and 52,049 admissions for the two cohorts. In this data set, procedures and medicines are grouped into intervention codes, together with diagnosis codes forming a patient's admission record. The number of diagnosis and intervention codes are 249 and 1071, respectively. We follow the same preprocessing steps and data split as in Pham et al., (2017). Different from MIMIC-III, a patient record suffering from chronic conditions often consists of multiple admissions, which is suitable for the task of predicting disease progression. The average number and the maximum number of admission per patient are 5.35 and 253, respectively.

Table 4.7: Example Recommended Medications by DMNCs on MIMIC-III dataset. Bold denotes matching against ground-truth.

| | |
|---|---|
| Diagnoses | Calculus Of Gallbladder (57411),Vascular disorders of male genital organs (60883), Abdominal Pain (78901), Poisoning By Other Tranquilizers (9695), Acute Myocardial Infarction Of Other Inferior Wall (41042), Hematoma Complicating (99812), Malignant hypertensive heart disease 40200), Dizziness and giddiness (7804), Venous (Peripheral) Insufficiency, Unspecified (45981), Hemorrhage Of Gastrointestinal Tract (5789) |
| Procedures | Coronary Bypass Of Three Coronary Arteries (3613), Single Internal Mammary Artery Bypass (3615), Extracorporeal circulation auxiliary to open heart surgery (3961), Insertion Of Intercostal Catheter For Drainage (3404), Operations on cornea(114) |
| Top 5 Ground-truth drugs (manually picked by experts) | Docusate Sodium (DOCU100L), Acetylsalicylic Acid (ASA81), Heparin (HEPA5I), Acetaminophen (ACET325), Potassium Chloride (KCLBASE2) |
| Top 5 Late-fusion Recommendations | **Docusate Sodium (DOCU100L)**, Neostigmine (NEOSI), **Acetaminophen (ACET325)**, Propofol (PROP100IG), **Potassium Chloride (KCLBASE2)** |
| Top 5 Early-fusion Recommendations | **Docusate Sodium (DOCU100L)**, **Acetaminophen (ACET325)**, **Potassium Chloride (KCLBASE2)**, Dextrose (DEX50SY), **Acetylsalicylic Acid (ASA81)** |
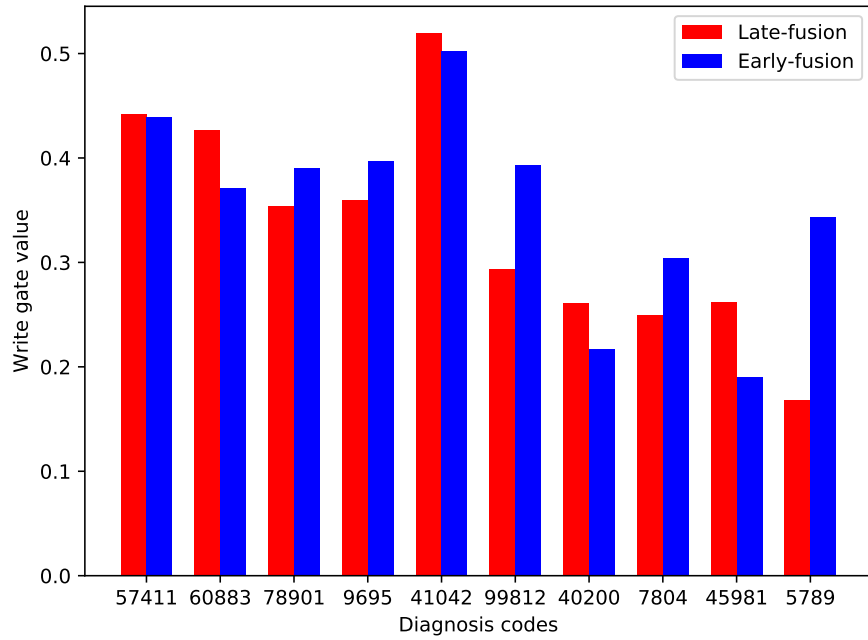
Figure 4.9: $M_1$'s $g_t^w$ over diagnoses. Diagnosis codes of a MIMIC-III patient is listed along the x-axis (ordered by priority) with the y-axis indicating how much the write gate allows a diagnosis to be written to the memory $M_1$.

Table 4.8: Regional hospital test results. P@K is precision at top K predictions in %.

| Model | Diabetes | | | Mental | | |
|---|---|---|---|---|---|---|
| | P@1 | P@2 | P@3 | P@1 | P@2 | P@3 |
| DeepCare | 66.2 | 59.6 | 53.7 | 52.7 | 46.9 | 40.2 |
| WLAS | 65.9 | 60.8 | 56.5 | 51.8 | 48.9 | 45.7 |
| $DMNC_l$ | 66.5 | **61.3** | **57.0** | 52.7 | 49.4 | 46.2 |
| $DMNC_e$ | **67.6** | 61.2 | 56.9 | **53.6** | **50.0** | **47.1** |

**Evaluations:** For comparison, we choose the second best-runner in our previous experiments WLAS and the current state-of-the-art DeepCare (Pham et al., 2017) as the two baselines.

**Implementations:** We use the validation data set to tune the hyper-parameters of our implementing methods and have the best embedding and hidden dimensions are 20 and 64, respectively. The word and memory size for DMNC are found to be 32 and 32, respectively. For performance measurements, we use P@$k$ metric ($k = 1, 2, 3$) to make it comparable with DeepCare's results reported in Pham et al., (2017).
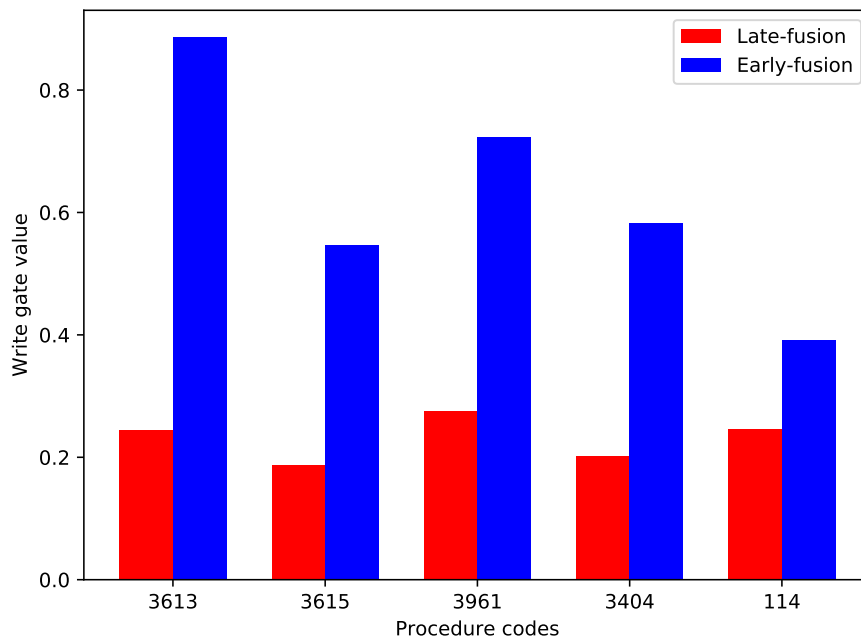
Figure 4.10: $M_2$'s $g_t^w$ over procedures. Medical procedure codes of a MIMIC-III patient is listed along the x-axis (in the order of executions) with the y-axis indicating how much the write gate allows a procedure to be written to the memory $M_2$.

**Results:** We report the results on test data of models for disease progression task in Table 4.8. For both cohorts, our proposed model consistently outperforms other methods and the performance gains become larger as the number of predictions increase. Compared to DeepCare which uses pre-trained embeddings and time-intervals as extra information, our methods only use raw medical codes and perform better. This emphasises the importance of modeling view interactions at event level. The late-fusion DMNC seems to perform slightly better than the early-fusion DMNC in the diabetes cohort, yet overall, the latter is the better one, which again validates its ability to model all types of view interactions.

## 4.6 Closing Remarks

In this chapter, we have introduced DCw-MANN and DMNC, which are slot-based MANNs designed for multiple processes. Under our designs, each input sequence is assigned a neural controller to encode and store its events to a dedicated memory. After all input sequences are stored, a decoder will access the memories and synthesise the read contents to produce the final output. Our methods can be generalised to sequence-to-sequence and multi-view prediction tasks that require special

handling of long-term dependencies and view interactions.

In summary, our main contributions are: (i) handling very long-term dependencies and rare events in healthcare data by solving the sequence prediction problem, (ii) proposing a novel memory-augmented architecture that uses dual controller and write-protected mechanism (DCw-MANN) to fit with sequence-in-sequence-out (SISO) task, (iii) proposing a novel dual memory neural computer (DMNC) to solve the asynchronous multi-view sequential problem and designing our architecture to model view interactions and long-term dependencies, (iv) demonstrating the efficacy of our proposed model on real-world medical data sets for the problems of treatment recommendation, drug prescription and disease progression. In particular, our models outperform other baselines by 1-2% across various metrics, which is significant in the domain. More importantly, explainability is critical in healthcare and we can somehow explain the behavior of our models by analysing the write gate of the memory.

We wish to emphasise that although our models are designed as predictive model targeted to healthcare, they can be applied to other sequential domains with similar data characteristics (i.e., sequential, long-term, rare and multi-view) such as video understanding. Also, the current work is limited to generating simple and deterministic output sequences. In the next chapter, we will focus on another important problem, in which, an external memory is necessary for holding temporal information and composing mixture distributions in the latent space of generative models.

# Chapter 5

# Variational Memory in Generative Models

## 5.1 Introduction

In the previous chapter, we have addressed the problem of encoder-decoder architecture with long-term dependencies. However, the decoding is an ill-posed problem, where there are many possible decoded sequences given an input sequence. To account for such variation, we need a method to model latent variables underlying these uncertainty. This lends naturally to generative model of sequences.

Recent advances in generative modeling have led to exploration of generative tasks. While generative models such as GAN (Goodfellow et al., 2014) and VAE (Kingma and Welling, 2014; Rezende et al., 2014) have been applied successfully for image generation, learning generative models for sequential discrete data is a long-standing problem. Early attempts to generate sequences using RNNs (Graves, 2013) and neural encoder-decoder models (Kalchbrenner and Blunsom, 2013; Vinyals and Le, 2015) gave promising results, but the deterministic nature of these models proves to be inadequate in many realistic settings. Tasks such as translation, question-answering and dialog generation would benefit from stochastic models that can produce a variety of outputs for an input. For example, there are several ways to translate a sentence from one language to another, multiple answers to a question and multiple responses for an utterance in conversation.

For tasks involving language understanding and production, handling intrinsic un-

certainty and latent variations is necessary. The choice of words and grammars may change erratically depending on speaker intentions, moods and previous languages used. The underlying RNNs in neural sequential models find it hard to capture the dynamics and their outputs are often trivial or too generic (Li et al., 2016). One way to overcome these problems is to introduce variability into these models. Unfortunately, sequential data such as speech and natural language is a hard place to inject variability (Serban et al., 2017) since they require a coherence of grammars and semantics yet allow freedom of word choice.

We propose a novel hybrid approach that integrates MAED (Sec. 3.3.5) and VAE, called Variational Memory Encoder-Decoder (VMED), to model the sequential properties and inject variability in sequence generation tasks. In this proposal, we utilise DC-MANN described in Sec. 4.3 where the powerful DNC (Graves et al., 2016) is chosen as the external memory. We prefer to allow writing to the memory during inference because in this work, we focus on generating diverse output sequences, which requires a dynamic memory for both encoding and decoding process. Furthermore, we introduce latent random variables to model the variability observed in the data and capture dependencies between the latent variables across timesteps. Our assumption is that there are latent variables governing an output at each timestep. In the conversation context, for instance, the latent space may represent the speaker's hidden intention and mood that dictate word choice and grammars. For a rich latent multimodal space, we use a Mixture of Gaussians (MoG) because a spoken word's latent intention and mood can come from different modes, e.g., whether the speaker is asking or answering, or she/he is happy or sad. By modeling the latent space as an MoG where each mode associates with some memory slot, we aim to capture multiple modes of the speaker's intention and mood when producing a word in the response. Since the decoder in our model has multiple read heads, the MoG can be computed directly from the content of chosen memory slots. Our external memory plays a role as a mixture model distribution generating the latent variables that are used to produce the output and take part in updating the memory for future generative steps.

To train our model, we adapt Stochastic Gradient Variational Bayes (SGVB) framework (Kingma and Welling, 2014). Instead of minimising the $KL$ divergence directly, we resort to using its variational approximation (Hershey and Olsen, 2007) to accommodate the MoG in the latent space. We show that minimising the approximation results in $KL$ divergence minimisation. We further derive an upper bound on our total timestep-wise $KL$ divergence and demonstrate that minimising the upper bound

is equivalent to fitting a continuous function by a scaled MoG. We validate the proposed model on the task of conversational response generation. This task serves as a nice testbed for the model because an utterance in a conversation is conditioned on previous utterances, the intention and the mood of the speaker. Finally, we evaluate our model on two open-domain and two closed-domain conversational datasets. The results demonstrate our proposed VMED gains significant improvement over state-of-the-art alternatives.

## 5.2 Preliminaries

### 5.2.1 Conditional Variational Autoencoder (CVAE) for Conversation Generation

A dyadic conversation can be represented via three random variables: the conversation context $x$ (all the chat before the response utterance), the response utterance $y$ and a latent variable $z$, which is used to capture the latent distribution over the reasonable responses. A variational autoencoder conditioned on $x$ (CVAE) is trained to maximise the conditional log likelihood of $y$ given $x$, which involves an intractable marginalisation over the latent variable $z$, i.e.,

$$p\left(y \mid x\right) = \int_z p\left(y, z \mid x\right) dz = \int_z p\left(y \mid x, z\right) p\left(z \mid x\right) dz \tag{5.1}$$

Fortunately, CVAE can be efficiently trained with the Stochastic Gradient Variational Bayes (SGVB) framework (Kingma and Welling, 2014) by maximising the variational lower bound of the conditional log likelihood. In a typical CVAE work, $z$ is assumed to follow multivariate Gaussian distribution with a diagonal covariance matrix, which is conditioned on $x$ as $p_\phi\left(z \mid x\right)$ and a recognition network $q_\theta(z \mid x, y)$ to approximate the true posterior distribution $p(z \mid x, y)$. The variational lower bound becomes

$$L\left(\phi, \theta; y, x\right) = -KL\left(q_\theta\left(z \mid x, y\right) \parallel p_\phi\left(z \mid x\right)\right) \tag{5.2}$$

$$+ \mathbb{E}_{q_\theta(z|x,y)}\left[\log p\left(y \mid x, z\right)\right] \leq \log p\left(y \mid x\right) \tag{5.3}$$

where $KL$ is the Kullback–Leibler divergence. With the introduction of the neural approximator $q_\theta(z \mid x, y)$ and the reparameterisation trick (Kingma et al., 2014), we can apply the standard back-propagation to compute the gradient of the variational lower bound. Fig. 5.1(a) depicts elements of the graphical model for this approach in the case of using CVAE.

## 5.2.2   Related Works

With the recent revival of recurrent neural networks (RNNs), there has been much effort spent on learning generative models of sequences. Early attempts include training RNN to generate the next output given previous sequence, demonstrating RNNs' ability to generate text and handwriting images (Graves, 2013). Later, encoder-decoder architecture (Sutskever et al., 2014b) enables generating a whole sequence in machine translation (Kalchbrenner and Blunsom, 2013), text summation (Nallapati et al., 2016) and conversation generation (Vinyals and Le, 2015). Although these models have achieved significant empirical successes, they fall short to capture the complexity and variability of sequential processes.

These limitations have recently triggered a considerable effort on introducing variability into the encoder-decoder architecture. Most of the methods focus on conditional VAE (CVAE) by constructing a variational lower bound conditioned on the context. The setting can be found in many applications including machine translation (Zhang et al., 2016a) and dialog generation (Bowman et al., 2016; Serban et al., 2017; Shen et al., 2017; Zhao et al., 2017). A common trick is to place a neural net between the encoder and the decoder to compute the Gaussian prior and posterior of the CVAE. This design is further enhanced by the use of external memory (Chen et al., 2018) and reinforcement learning (Wen et al., 2017). In contrast to this design, our VMED uses recurrent latent variable approach (Chung et al., 2015), that is, our model requires a CVAE for each step of generation. Besides, our external memory is used for producing the latent distribution, which is different from the one proposed in (Chen et al., 2018) where the memory is used only for holding long-term dependencies at sentence level. Compared to variational addressing scheme mentioned in (Bornschein et al., 2017), our memory uses deterministic addressing scheme, yet the memory content itself is used to introduce randomness to the architecture. More relevant to our work is GTMM (Gemici et al., 2017) where memory read-outs involve in constructing the prior and posterior at every timestep. However, this approach uses Gaussian prior without conditional context.
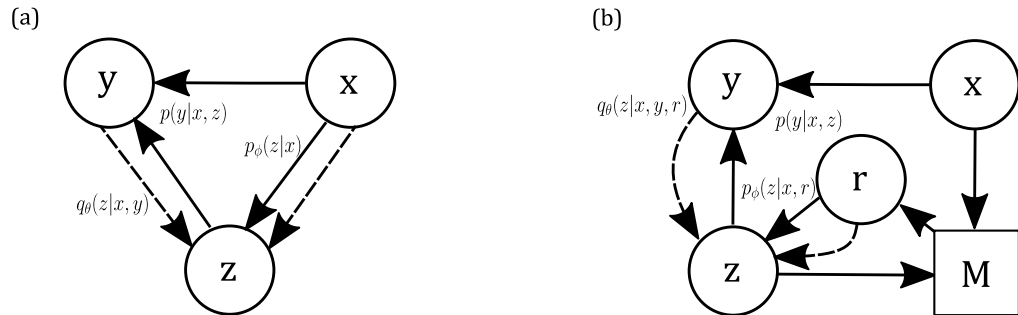
Figure 5.1: Graphical Models of the vanilla CVAE (a) and our proposed VMED (b)

Using mixture of models instead of single Gaussian in VAE framework is not a new concept. Several works proposed replacing the Gaussian prior and posterior in VAE by MoGs for clustering and generating image problems (Dilokthanakul et al., 2016; Jiang et al., 2017; Nalisnick et al., 2016). Others applied MoG prior to model transitions between video frames and caption generation (Shu et al., 2016; Wang et al., 2017). These methods use simple feed forward network to produce Gaussian sub-distributions independently. In our model, on the contrary, memory slots are strongly correlated with each others, and thus modes in our MoG work together to define the shape of the latent distributions at specific timestep. To the best of our knowledge, our work is the first attempt to use an external memory to induce mixture models for sequence generation problems.

## 5.3 Variational Memory Encoder-Decoder

Built upon CVAE and partly inspired by VRNN (Chung et al., 2015), we introduce a novel memory-augmented variational recurrent network dubbed Variational Memory Encoder-Decoder (VMED). With an external memory module, VMED explicitly models the dependencies between latent random variables across subsequent timesteps. However, unlike the VRNN which uses hidden values of RNN to model the latent distribution as a Gaussian, our VMED uses read values $r$ from an external memory $M$ as a Mixture of Gaussians (MoG) to model the latent space. This choice of MoG also leads to new formulation for the prior $p_\phi$ and the posterior $q_\theta$ mentioned in Eq. (5.2). The graphical representation of our model is shown in Fig. 5.1(b).

## 5.3.1 Generative Process

The VMED includes a CVAE at each time step of the decoder. These CVAEs are conditioned on the context sequence via $K$ read values $r_{t-1} = \left[r_{t-1}^1, r_{t-1}^2, ..., r_{t-1}^K\right]$ from the external memory. Since the read values are conditioned on the previous state of the decoder $h_{t-1}^d$, our model takes into account the temporal structure of the output. Unlike other designs of CVAE where there is often only one CVAE with a Gaussian prior for the whole decoding process, our model keeps reading the external memory to produce the prior as a Mixture of Gaussians at every timestep. At the $t$-th step of generating an utterance in the output sequence, the decoder will read from the memory $K$ read values, representing $K$ modes of the MoG. This multi-modal prior reflects the fact that given a context $x$, there are different modes of uttering the output word $y_t$, which a single mode cannot fully capture. The MoG prior distribution is modeled as

$$g_t = p_\phi\left(z_t \mid x, r_{t-1}\right) = \sum_{i=1}^{K} \pi_t^{i,x}\left(x, r_{t-1}^i\right) \mathcal{N}\left(z_t; \mu_t^{i,x}\left(x, r_{t-1}^i\right), \sigma_t^{i,x}\left(x, r_{t-1}^i\right)^2 \mathbf{I}\right) \quad (5.4)$$

We treat the mean $\mu_t^{i,x}$ and standard deviation (s.d.) $\sigma_t^{i,x}$ of each Gaussian distribution in the prior as neural functions of the context sequence $x$ and read vectors from the memory. The context is encoded into the memory by an $LSTM^E$ encoder. In decoding, the decoder $LSTM^D$ attends to the memory and choose $K$ read vectors. We split each read vector into two parts $r^{i,\mu}$ and $r^{i,\sigma}$, each of which is used to compute the mean and s.d., respectively: $\mu_t^{i,x} = r_{t-1}^{i,\mu}$, $\sigma_t^{i,x} = softplus\left(r_{t-1}^{i,\sigma}\right)$. Here we use the softplus function for computing s.d. to ensure the positiveness. The mode weight $\pi_t^{i,x}$ is chosen based on the read attention weights $w_{t-1}^{i,r}$ over memory slots. Since we use soft-attention, a read value is computed from all slots yet the main contribution comes from the one with highest attention score. Thus, we pick the maximum attention score in each read weight and normalise to become the mode weights: $\pi_t^{i,x} = \max w_{t-1}^{i,r} / \sum_{i=1}^{i=K} \max w_{t-1}^{i,r}$.

Armed with the prior, we follow a recurrent generative process by alternatively using the memory to compute the MoG and using latent variable $z$ sampled from the MoG to update the memory and produce the output conditional distribution. The pseudo-algorithm of the generative process is given in Algorithm 5.1.

---

**Algorithm 5.1** VMED Generation

---

**Require:** Given $p_\phi$, $\left[r_0^1, r_0^2, ..., r_0^K\right]$, $h_0^d$, $y_0^*$

1: **for** $t = 1, T$ **do**
2:       Sampling $z_t \sim p_\phi\left(z_t \mid x, r_{t-1}\right)$ in Eq.(5.4)
3:       Compute: $o_t^d, h_t^d = LSTM^D\left(\left[y_{t-1}^*, z_t\right], h_{t-1}^d\right)$
4:       Compute the conditional distribution: $p\left(y_t \mid x, z_{\leq t}\right) = \text{softmax}\left(W_{out} o_t^d\right)$
5:       Update memory and read $[r_t^1, r_t^2, ..., r_t^K]$ using $h_t^d$ as in DNC
6:       Generate output $y_t^* = \underset{y \in Vocab}{argmax}\, p\left(y_t = y \mid x, z_{\leq t}\right)$
7: **end for**

---

## 5.3.2 Neural Posterior Approximation

At each step of the decoder, the true posterior $p\left(z_t \mid x, y\right)$ will be approximated by a neural function of $x, y$ and $r_{t-1}$, denoted as $q_\theta\left(z_t \mid x, y, r_{t-1}\right)$ . Here, we use a Gaussian distribution to approximate the posterior. The unimodal posterior is chosen because given a response $y$, it is reasonable to assume only one mode of latent space is responsible for this response. Also, choosing a unimodel will allow the reparameterisation trick during training and reduce the complexity of $KL$ divergence computation. The approximated posterior is computed by the following the equation

$$f_t = q_\theta\left(z_t \mid x, y_{\leq t}, r_{t-1}\right) = \mathcal{N}\left(z_t; \mu_t^{x,y}\left(x, y_{\leq t}, r_{t-1}\right), \sigma_t^{x,y}\left(x, y_{\leq t}, r_{t-1}\right)^2 \mathbf{I}\right) \qquad (5.5)$$

with mean $\mu_t^{x,y}$ and s.d. $\sigma_t^{x,y}$. We use an $LSTM^U$ utterance encoder to model the ground truth utterance sequence up to timestep $t$-th $y_{\leq t}$. The $t$-th hidden value of the $LSTM^U$ is used to represent the given data in the posterior: $h_t^u = LSTM^U\left(y_t, h_{t-1}^u\right)$. The neural posterior combines the read values $\mathbf{r}_t = \sum_{i=1}^{K} \pi_t^{i,x} r_{t-1}^i$ together with the ground truth data to produce the Gaussian posterior: $\mu_t^{x,y} = W_\mu\left[\mathbf{r}_t, h_t^u\right], \sigma_t^{x,y} = softplus\left(W_\sigma\left[\mathbf{r}_t, h_t^u\right]\right)$. In these equations, we use learnable matrix weights $W_\mu$ and $W_\sigma$ as a recognition network to compute the mean and s.d. of the posterior, ensuring that the distribution has the same dimension as the prior. We apply the reparamterisation trick to calculate the random variable sampled from the posterior as $z_t' = \mu_t^{x,y} + \sigma_t^{x,y} \odot \epsilon$, $\epsilon \in \mathcal{N}\left(0, \mathbf{I}\right)$. Intuitively, the reparameterisation trick bridges the gap between the generation model and the inference model during the training.

### 5.3.3 Learning

In the training phase, the neural posterior is used to produce the latent variable $z'_t$. The read values from memory are used directly as the MoG priors and the priors are trained to approximate the posterior by reducing the $KL$ divergence. During testing, the decoder uses the prior for generating latent variable $z_t$, from which the output is computed. The training and testing diagram is illustrated in Fig. 5.2. The objective function becomes a timestep-wise variational lower bound by following similar derivation presented in Chung et al., (2015),

$$\mathcal{L}(\theta, \phi; y, x) = E_{q*} \left[ \sum_{t=1}^{T} -KL\left(q_\theta\left(z_t \mid x, y_{\leq t}, r_{t-1}\right) \parallel p_\phi\left(z_t \mid x, r_{t-1}\right)\right) + \log p\left(y_t \mid x, z_{\leq t}\right) \right]$$
(5.6)

where $q* = q_\theta\left(z_{\leq T} \mid x, y_{\leq T}, r_{<T}\right)$. To maximise the objective function, we have to compute $KL$ divergence between $f_t = q_\theta\left(z_t \mid x, y_{\leq t}, r_{t-1}\right)$ and $g_t = p_\phi\left(z_t \mid x, r_{t-1}\right)$. Since there is no closed-form for this $KL\left(f_t \parallel g_t\right)$ between Gaussian $f_t$ and Mixture of Gaussians $g_t$, we use a closed-form approximation named $D_{var}$ (Hershey and Olsen, 2007) to replace the $KL$ term in the objective function. For our case: $KL\left(f_t \parallel g_t\right) \approx D_{var}\left(f_t \parallel g_t\right) = -\log \sum_{i=1}^{K} \pi^i e^{-KL\left(f_t \parallel g_t^i\right)}$. Here, $KL\left(f_t \parallel g_t^i\right)$ is the $KL$ divergence between two Gaussians and $\pi^i$ is the mode weight of $g_t$. The final objective function is

$$
\begin{aligned}
\mathcal{L} = &\sum_{t=1}^{T} \log \sum_{i=1}^{K} \left[ \pi_t^{i,x} \exp\left(-KL\left(\mathcal{N}\left(\mu_t^{x,y}, \sigma_t^{x,y2}\mathbf{I}\right) \parallel \mathcal{N}\left(\mu_t^{i,x}, \sigma_t^{i,x2}\mathbf{I}\right)\right)\right) \right] \\
&+ \frac{1}{L}\sum_{t=1}^{T}\sum_{l=1}^{L} \log p\left(y_t \mid x, z_{\leq t}^{(l)}\right)
\end{aligned}
$$
(5.7)

### 5.3.4 Theoretical Analysis

We now show that by modeling the prior as MoG and the posterior as Gaussian, minimising the approximation results in $KL$ divergence minimisation.

**Theorem 5.1.** *The KL divergence between a Gaussian and a Mixture of Gaussians has an upper bound $D_{var}$.*
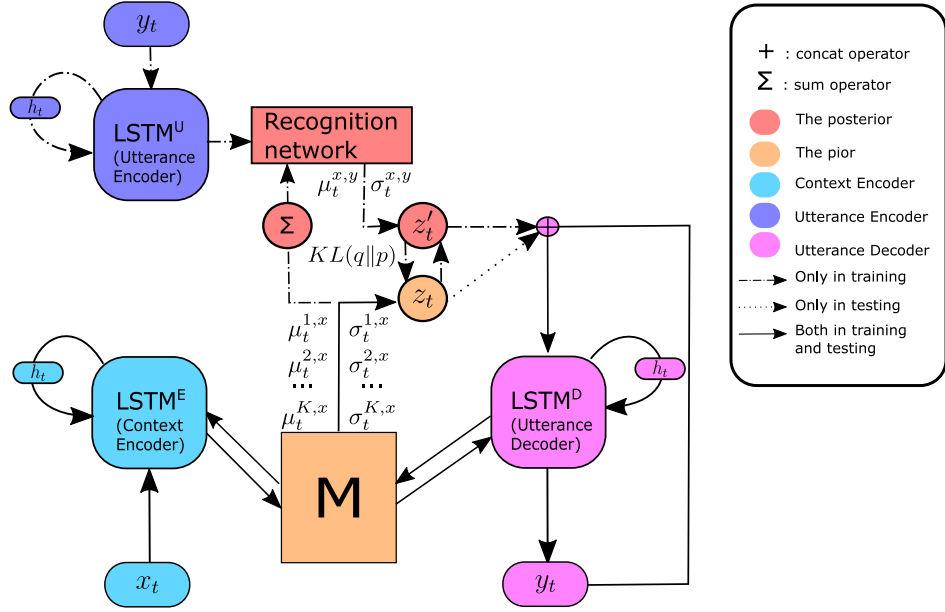
Figure 5.2: Training and testing of VMED

A sketch of the proof is derived (see C.1 for full derivation). Let define the log-likelihood $L_f(g) = E_{f(x)}[\log g(x)]$, we have

$$L_f(g) \geq \log \sum_{i=1}^{K} \pi^i e^{-KL(f\|g^i)} + L_f(f) = -D_{var} + L_f(f)$$

$$\Rightarrow D_{var} \geq L_f(f) - L_f(g) = KL(f \| g)$$

Thus, minimising $D_{var}$ results in $KL$ divergence minimisation. Next, we establish an upper bound on the total timestep-wise $KL$ divergence in Eq. (5.6) and show that minimising this upper bound is equivalent to fitting a continuous function by a scaled MoG. The total timestep-wise $KL$ divergence reads

$$\sum_{t=1}^{T} KL(f_t \| g_t) = \int_{-\infty}^{+\infty} \sum_{t=1}^{T} f_t(x) \log[f_t(x)] \, dx \quad - \int_{-\infty}^{+\infty} \sum_{t=1}^{T} f_t(x) \log[g_t(x)] \, dx$$

where $g_t = \sum_{i=1}^{K} \pi_t^i g_t^i$ and $g_t^i$ is the $i$-th Gaussian in the MoG at timestep $t$-th. If at each decoding step, minimising $D_{var}$ results in adequate $KL$ divergence such that the prior is optimised close to the neural posterior, according to Chebyshev's sum

Table 5.1: BLEU-1, 4 and A-Glove on testing datasets. B1, B4, AG are acronyms for BLEU-1, BLEU-4, A-Glove metrics, respectively (higher is better).

| Model | Cornell Movies | | | OpenSubtitle | | | LJ users | | | Reddit comments | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | B1 | B4 | AG | B1 | B4 | AG | B1 | B4 | AG | B1 | B4 | AG |
| Seq2Seq | 18.4 | 9.5 | 0.52 | 11.4 | 5.4 | 0.29 | 13.1 | 6.4 | 0.45 | 7.5 | 3.3 | 0.31 |
| Seq2Seq-att | 17.7 | 9.2 | 0.54 | 13.2 | 6.5 | 0.42 | 11.4 | 5.6 | 0.49 | 5.5 | 2.4 | 0.25 |
| DNC | 17.6 | 9.0 | 0.51 | 14.3 | 7.2 | 0.47 | 12.4 | 6.1 | 0.47 | 7.5 | 3.4 | 0.28 |
| CVAE | 16.5 | 8.5 | 0.56 | 13.5 | 6.6 | 0.45 | 12.2 | 6.0 | 0.48 | 5.3 | 2.8 | 0.39 |
| VLSTM | 18.6 | 9.7 | 0.59 | 16.4 | 8.1 | 0.43 | 11.5 | 5.6 | 0.46 | 6.9 | 3.1 | 0.27 |
| VMED (K=1) | 20.7 | 10.8 | 0.57 | 12.9 | 6.2 | 0.44 | 13.7 | 6.9 | 0.47 | 9.1 | 4.3 | 0.39 |
| VMED (K=2) | 22.3 | 11.9 | **0.64** | 15.3 | 8.8 | 0.49 | 15.4 | 7.9 | **0.51** | 9.2 | 4.4 | 0.38 |
| VMED (K=3) | 19.4 | 10.4 | 0.63 | **24.8** | **12.9** | **0.54** | **18.1** | **9.8** | 0.49 | **12.3** | **6.4** | **0.46** |
| VMED (K=4) | **23.1** | **12.3** | 0.61 | 17.9 | 9.3 | 0.52 | 14.4 | 7.5 | 0.47 | 8.6 | 4.6 | 0.41 |

inequality, we can derive an upper bound on the total timestep-wise $KL$ divergence as (see Supplementary Materials for full derivation)

$$\int_{-\infty}^{+\infty} \sum_{t=1}^{T} f_t(x) \log \left[ f_t(x) \right] dx \quad - \int_{-\infty}^{+\infty} \frac{1}{T} \sum_{t=1}^{T} f_t(x) \log \left[ \prod_{t=1}^{T} g_t(x) \right] dx \qquad (5.8)$$

The left term is sum of the entropies of $f_t(x)$, which does not depend on the training parameter $\phi$ used to compute $g_t$, so we can ignore that. Thus given $f$, minimising the upper bound of the total timestep-wise $KL$ divergence is equivalent to maximising the right term of Eq. (5.8). Since $g_t$ is an MoG and products of MoG is proportional to an MoG, $\prod_{t=1}^{T} g_t(x)$ is a scaled MoG (see Supplementary material for full proof). Maximising the right term is equivalent to fitting function $\sum_{t=1}^{T} f_t(x)$, which is sum of Gaussians and thus continuous, by a scaled MoG. This, in theory, is possible regardless of the form of $f_t$ since MoG is a universal approximator (Bacharoglou, 2010; Maz'ya and Schmidt, 1996).

# 5.4 Experiments and Results

## 5.4.1 Quantitative Results

**Datasets and pre-processing**: We perform experiments on two collections: The first collection includes open-domain movie transcript datasets containing casual conversations: Cornell Movies[1] and OpenSubtitle[2]. They have been used commonly in evaluating conversational agents (Lison and Bibauw, 2017; Vinyals and Le, 2015). The second are closed-domain datasets crawled from specific domains, which are question-answering of LiveJournal (LJ) users and Reddit comments on movie topics. For each dataset, we use 10,000 conversations for validating and 10,000 for testing.

**Baselines, implementations and metrics**: We compare our model with three deterministic baselines: the encoder-decoder neural conversational model (Seq2Seq) (Vinyals and Le, 2015) and its two variants equipped with attention mechanism (Cho et al., 2014a; Bahdanau et al., 2015) (Seq2Seq-att) and a DNC external memory (Graves et al., 2016) (DNC). The vanilla CVAE is also included in the baselines. To build this CVAE, we follow similar architecture introduced in (Zhao et al., 2017) without bag-of-word loss and dialog act features[3]. A variational recurrent model without memory is also included in the baselines. The model termed VLSTM is implemented based on LSTM instead of RNN as in VRNN framework (Chung et al., 2015). We try our model VMED[4] with different number of modes ($K = 1, 2, 3, 4$). It should be noted that, when $K = 1$, our model's prior is exactly a Gaussian and the $KL$ term in Eq. (7.6) is no more an approximation. Details of dataset descriptions and model implementations are included in Supplementary material.

We report results using two performance metrics in order to evaluate the system from various linguistic points of view: (i) Smoothed Sentence-level BLEU (Chen and Cherry, 2014): BLEU is a popular metric that measures the geometric mean of modified ngram precision with a length penalty. We use BLEU-1 to 4 as our lexical similarity. (ii) Cosine Similarly of Sentence Embedding: a simple method to obtain sentence embedding is to take the average of all the word embeddings in the sentences (Forgues et al., 2014). We follow Zhao et al., (2017) and choose Glove

---

[1] http://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html

[2] http://opus.nlpl.eu/OpenSubtitles.php

[3] Another variant of non-memory CVAE with MoG prior is also examined. We produce a set of MoG parameters by a feed forward network with the input as the last encoder hidden states. However, the model is hard to train and fails to converge with these datasets.

[4] Source code is available at https://github.com/thaihungle/VMED

(Levy and Goldberg, 2014) as the word embedding in measuring sentence similarly (A-Glove) (Zhao et al., 2017). To measure stochastic models, for each input, we generate output ten times. The metric between the ground truth and the generated output is calculated and taken average over ten responses.

**Metric-based Analysi**s: We report results on four test datasets in Table 5.1. For BLEU scores, here we only list results for BLEU-1 and 4. Other BLEUs show similar pattern and will be listed in Supplementary material. As clearly seen, VMED models outperform other baselines over all metrics across four datasets. In general, the performance of Seq2Seq is comparable with other deterministic methods despite its simplicity. Surprisingly, CVAE or VLSTM does not show much advantage over deterministic models. As we shall see, although CVAE and VLSTM responses are diverse, they are often out of context. Amongst different modes of VMED, there is often one best fit with the data and thus shows superior performance. The optimal number of modes in our experiments often falls to $K = 3$, indicating that increasing modes does not mean to improve accuracy.

It should be noted that there is inconsistency between BLEU scores and A-Glove metrics. This is because BLEU measures lexicon matching while A-Glove evaluates semantic similarly in the embedding space. For example, two sentences having different words may share the same meaning and lie close in the embedding space. In either case, compared to others, our optimal VMED always achieves better performance.

## 5.4.2 Qualitative Analysis

Table 5.2 represents responses generated by experimental models in reply to different input sentences. The replies listed are chosen randomly from 50 generated responses whose average of metric scores over all models are highest. For stochastic models, we generate three times for each input, resulting in three different responses. In general, the stochastic models often yield longer and diverse sequences as expected. For closed-domain cases, all models responses are fairly acceptable. Compared to the rest, our VMED's responds seem to relate more to the context and contain meaningful information. In this experiment, the open-domain input seems nosier and harder than the closed-domain ones, thus create a big challenge for all models. Despite that, the quality of VMED's responses is superior to others. Amongst deterministic models, DNC's generated responses look more reasonable than Seq2Seq's

Table 5.2: Examples of context-response pairs. /*/ denotes separations between stochastic responses.

| Input context | Response |
|---|---|
| **Reddit comment:** What is your favorite scene in film history ? Mine is the restaurant scene in the Godfather. | **Seq2Seq:** The scene in<br>**Seq2Seq-att:** The final<br>**DNC:** The scene in<br>**CVAE:** Inception god! Not by a shark /*/ Amour great /*/ Pro thing you know 3 dead<br>**VLSTM:** The scene in /*/ The of a dead /*/ The sky in scene<br>**VMED (K=3):** The opening scene from history movie /*/ The scene in a shot nights! Robin movie /*/ The psycho scene in fight from |
| **Reddit comment:** What actor will win an Oscar in the next 10 years ? | **Seq2Seq:** Colin<br>**Seq2Seq-att:** Liam Neeson<br>**DNC:** Tom Gyllenhaal<br>**CVAE:** Daryl and Aaron /*/ Carefully count Alfred Deniro /*/ Ponyo Joker possible<br>**VLSTM:** Michael Bullock /*/ Michael /*/ Michael De<br>**VMED (K=3):** Edward or Leo Dicaprio goes on /*/ Dicaprio will /*/ Dicaprio Tom has actually in jack on road |
| **LJ user:** Your very pretty and your is very cute. May i add ya to my friends list ? | **Seq2Seq:** I! Add and too you back<br>**Seq2Seq-att:** I did may! Away back<br>**DNC:** Sure you added<br>**CVAE:** Sure ! I can add you /*/ Thank cool of surely /*/ Thank hi of sure!<br>**VLSTM:** Haha of do go /*/ Haha of sure! Yes lol /*/ Haha sure! Go<br>**VMED (K=3):** Thank cool for sure /*/ Add definitely! Sure surely friends /*/ Thank girl go ahead |
| **Cornell movies:** Nobody can murder someone with their feelings . I mean that is crazy ? | **Seq2Seq:** Yes you are<br>**Seq2Seq-att:** Really not is it<br>**DNC:** Managing the truth<br>**CVAE:** Sure not to be in here /*/ Oh yes but i know! /*/ That to doing with here and<br>**VLSTM:** I am dead . ! That is ... /*/ I did not what way . /*/ I am not . But his things with ...<br>**VMED (K=4):** You are right! /*/ That is crazy /*/ You can't know Jimmy |
| **OpenSubtitle:** I'm obliged for your hospitality. I appreciate it and your husband too. | **Seq2Seq:** That is have got coming about these<br>**Seq2Seq-att:** May you not what nothing nobody<br>**DNC:** Yes i am taking this<br>**CVAE:** No . that for good! And okay /*/ All in the of two thing /*/ Sure. Is this!<br>**VLSTM:** I ... /*/ I understand /*/ I ! . ...<br>**VMED (K=3):** I know. I can afford /*/ I know nothing to store for you pass /*/ I know. Doing anymore you father |

even though its BLEU scores are not always higher. Perhaps, the reference to external memory at every timestep enhances the coherence between output and input, making the response more related to the context. VMED may inherit this feature from its external memory and thus tends to produce reasonable responses. By contrast, although responses from CVAE and VLSTM are not trivial, they have more grammatical errors and sometimes unrelated to the topic.

## 5.5 Closing Remarks

In this chapter, we propose a novel approach to sequence generation called Variational Memory Encoder-Decoder (VMED) that introduces variability into encoder-decoder architecture via the use of external memory as mixture model. By modeling the latent temporal dependencies across timesteps, our VMED produces a MoG representing the latent distribution. Each mode of the MoG associates with some memory slot and thus captures some aspect of context supporting generation process. To accommodate the MoG, we employ a $KL$ approximation and we demonstrate that minimising this approximation is equivalent to minimising the $KL$ divergence. We derive an upper bound on our total timestep-wise $KL$ divergence and indicate that the optimisation of this upper bound is equivalent to fitting a continuous function by an scaled MoG, which is in theory possible regardless of the function form. This forms a theoretical basis for our model formulation using MoG prior for every step of generation. We apply our proposed model to conversation generation problem. The results demonstrate that VMED outperforms recent advances both quantitatively and qualitatively.

So far we have designed and applied MANNs to long-term sequential modeling problems without considering the memorisation ability of these models. Besides, we have followed common memory access patterns without judging the effectiveness of these accesses. We will tackle these issues in the next chapter.

# Chapter 6

# Optimal Writing Memory

## 6.1 Introduction

Recall that a core task in sequence learning is to capture long-term dependencies amongst timesteps which demands memorisation of distant inputs (Sec. 2.4). In recurrent neural networks (RNNs), the memorisation is implicitly executed via integrating the input history into the state of the networks. However, learning vanilla RNNs over long distance proves to be difficult due to the vanishing gradient problem (Sec. 2.4.1). One alleviation is to introduce skip-connections along the execution path, in the form of dilated layers (Van Den Oord et al., 2016; Chang et al., 2017), which is not our main focus. Other solutions have been mentioned in Chapter 3 including attention mechanisms (Cho et al., 2014a; Vaswani et al., 2017; Bahdanau et al., 2015) and external slot-based memory (Graves et al., 2014, 2016).

Amongst all, using external memory most resembles human cognitive architecture where we perceive the world sequentially and make decision by consulting our memory. Recent attempts have simulated this process by using RAM-like memory architectures that store information into memory slots. Reading and writing are governed by neural controllers using attention mechanisms (Sec. 3.3).

Despite the promising empirical results, there is no theoretical analysis or clear understanding on optimal operations that a memory should have to maximise its performance. To the best of our knowledge, no solution has been proposed to help MANNs handle ultra-long sequences given limited memory. This scenario is practical because (i) sequences in the real-world can be very long while the computer resources

are limited and (ii) it reflects the ability to compress in human brain to perform life-long learning. Previous attempts (Rae et al., 2016) try to learn ultra-long sequences by expanding the memory, which is not always feasible and do not aim to optimise the memory by some theoretical criterion. This chapter presents a new approach towards finding optimal operations for MANNs that serve the purpose of learning longer sequences with finite memory.

More specifically, upon analysing RNN and MANN operations we first introduce a measurement on the amount of information that a MANN holds after encoding a sequence. This metric reflects the quality of memorisation under the assumption that contributions from timesteps are equally important. We then derive a generic solution to optimise the measurement. We term this optimal solution as Uniform Writing (UW), and it is applicable for any MANN due to its generality. Crucially, UW helps reduce significantly the computation time of MANN. Third, to relax the assumption and enable the method to work in realistic settings, we further propose Cached Uniform Writing (CUW) as an improvement over the Uniform Writing scheme. By combining uniform writing with local attention, CUW can learn to discriminate timesteps while maximising local memorisation. Finally we demonstrate that our proposed models outperform several MANNs and other state-of-the-art methods in various synthetic and practical sequence modeling tasks.

## 6.2 Related Backgrounds

Traditional recurrent models such as RNN/LSTM (Elman, 1990; Hochreiter and Schmidhuber, 1997) exhibit some weakness that prevents them from learning really long sequences. The reason is mainly due to the vanishing gradient problem or to be more specific, the exponential decay of input value over time, which is analysed in Sec. 2.4.1. It should be noted that although LSTM is designed to diminish the problem, it is limited to the capacity of cell memory. In other words, if the amount of relevant information exceeds the capacity, LSTM will forget distant inputs since exponential decay still exists outside the cell memory. One different approach to overcome this problem is enforcing the exponential decay factor close to one by putting a unitary constraint on the recurrent weight (Arjovsky et al., 2016; Wisdom et al., 2016). Although this approach is theoretically motivated, it restricts the space of learnt parameters.

More relevant to our work, the idea of using less or adaptive computation has been

proposed by many (Graves, 2016; Yu et al., 2017, 2018; Seo et al., 2018). Most of these works are based on the assumption that some of timesteps in a sequence are unimportant and thus can be ignored to reduce the cost of computation and increase the performance of recurrent networks. Different form our approach, these methods lack theoretical supports and do not directly aim to solve the problem of memorising long-term dependencies.

Dilated RNN (Chang et al., 2017) is another RNN-based proposal which improves long-term learning by stacking multiple dilated recurrent layers with hierarchical skip-connections. This theoretically guarantees the mean recurrent length and shares with our method the idea to construct a measurement on memorisation capacity of the system and propose solutions to optimise it. The difference is that our system is memory-augmented neural networks while theirs is multi-layer RNNs, which leads to totally different optimisation problems.

Recent researches recommend to replace traditional recurrent models by other neural architectures to overcome the vanishing gradient problem. The Transformer (Vaswani et al., 2017) attends to all timesteps at once, which ensures instant access to distant timestep yet requires quadratic computation and physical memory proportional to the sequence length (see Sec. 3.2.3). Memory-augmented neural networks (MANNs), on the other hand, learn to establish a limited-size memory and attend to the memory only, which is scalable to any-length sequence (see Sec. 3.3). Compared to others, MANNs resemble both computer architecture design and human working memory (Logie, 2014). However, the current understanding of the underlying mechanisms and theoretical foundations for MANNs are still limited.

Recent works on MANNs rely almost on reasonable intuitions. Some introduce new addressing mechanisms such as location-based (Graves et al., 2014), least-used (Santoro et al., 2016) and order-based (Graves et al., 2016). Others focus on the scalability of MANN by using sparse memory access to avoid attending to a large number of memory slots (Rae et al., 2016). These problems are different from ours which involves MANN memorisation capacity optimisation.

Our local optimal solution to this problem is related to some known neural caching (Grave et al., 2017b,a; Yogatama et al., 2018) in terms of storing recent hidden states for later encoding uses. These methods either aim to create structural bias to ease the learning process (Yogatama et al., 2018) or support large scale retrieval (Grave et al., 2017a). These are different from our caching purpose, which encourages overwriting and relaxes the equal contribution assumption of the optimal solution. Also,

the details of implementation are different as ours uses local memory-augmented attention mechanisms.

## 6.3 Theoretical Analysis on Memorisation

### 6.3.1 Generic Memory Operations

Memory-augmented neural networks can be viewed as an extension of RNNs with external memory $M$. The memory supports read and write operations based on the output $o_t$ of the controller, which in turn is a function of current timestep input $x_t$, previous hidden state $h_{t-1}$ and read value $r_{t-1}$ from the memory. Let assume we are given these operators from recent MANNs such as NTM (Graves et al., 2014) or DNC (Graves et al., 2016), represented as

$$r_t = f_r\left(o_t, M_{t-1}\right) \tag{6.1}$$

$$M_t = f_w\left(o_t, M_{t-1}\right) \tag{6.2}$$

The controller output and hidden state are updated as follows,

$$o_t = f_o\left(h_{t-1}, r_{t-1}, x_t\right) \tag{6.3}$$

$$h_t = f_h\left(h_{t-1}, r_{t-1}, x_t\right) \tag{6.4}$$

Here, $f_o$ and $f_h$ are often implemented as RNNs while $f_r$ and $f_w$ are designed specifically for different memory types.

Current MANNs only support regular writing by applying Eq. (6.2) every timestep. In effect, regular writing ignores the accumulated short-term memory stored in the controller hidden states which may well-capture the recent subsequence. We argue that the controller does not need to write to memory continuously as its hidden state also supports memorising. Another problem of regular writing is time complexity. As the memory access is very expensive, reading/writing at every timestep makes MANNs much slower than RNNs. This motivates a irregular writing strategy to utilise the memorisation capacity of the controller and consequently, speed up the

model. In the next sections, we first define a metric to measure the memorisation performance of RNNs, as well as MANNs. Then, we solve the problem of finding the best irregular writing that optimises the metric.

## 6.3.2 Memory Analysis of RNNs

We first define the ability to "remember" of recurrent neural networks, which is closely related to the vanishing/exploding gradient problem (Pascanu et al., 2013). In RNNs, the state transition $h_t = \phi\left(h_{t-1}, x_t\right)$ contains contributions from not only $x_t$, but also previous timesteps $x_{i<t}$ embedded in $h_{t-1}$. Thus, $h_t$ can be considered as a function of timestep inputs, i.e, $h_t = f\left(x_1, x_2, ..., x_t\right)$. One way to measure how much an input $x_i$ contributes to the value of $h_t$ is to calculate the norm of the gradient $\left\|\frac{\partial h_t}{\partial x_i}\right\|$. If the norm equals zero, $h_t$ is constant w.r.t $x_i$, that is, $h_t$ does not "remember" $x_i$. As a bigger $\left\|\frac{\partial h_t}{\partial x_i}\right\|$ implies more influence of $x_i$ on $h_t$, we propose using $\left\|\frac{\partial h_t}{\partial x_i}\right\|$ to measure the contribution of the $i$-th input to the $t$-th hidden state. Let $c_{i,t}$ denotes this term, we can show that in the case of common RNNs, $\lambda_c c_{i,t} \geq c_{i-1,t}$ with some $\lambda_c \in \mathbb{R}^+$(see Appendix D.1 - D.3 for proof). This means further to the past, the contribution decays (when $\lambda_c < 1$) or grows (when $\lambda_c > 1$) with the rate of at least $\lambda_c$ .We can measure the average amount of contributions across $T$ timesteps as follows (see Appendix D.4 for proof):

**Theorem 6.1.** *There exists $\lambda \in \mathbb{R}^+$ such that the average contribution of a sequence of length T with respect to a RNN can be quantified as the following,*

$$I_\lambda = \frac{\sum\limits_{t=1}^{T} c_{t,T}}{T} = c_{T,T}\frac{\sum\limits_{t=1}^{T} \lambda^{T-t}}{T} \tag{6.5}$$

If $\lambda < 1$, $\lambda^{T-t} \to 0$ as $T - t \to \infty$. This is closely related to vanishing gradient problem. LSTM is known to "remember" long sequences better than RNN by using extra memory gating mechanisms, which help $\lambda$ to get closer to 1. If $\lambda > 1$, the system may be unstable and suffer from the exploding gradient problem.

## 6.3.3 Memory Analysis of MANNs

In slot-based MANNs, memory $M$ is a set of $D$ memory slots. A write at step $t$ can be represented by the controller's hidden state $h_t$, which accumulates inputs
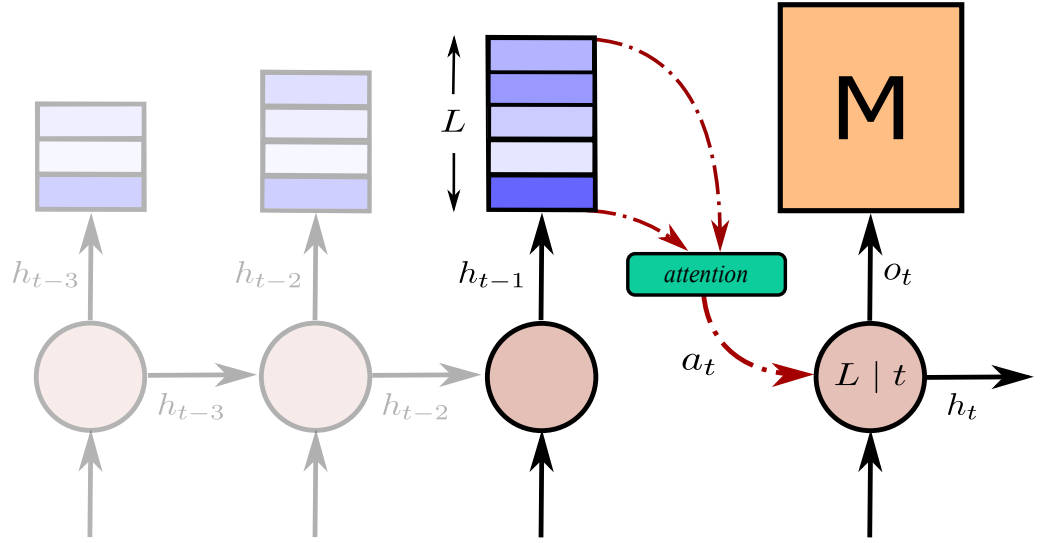
Figure 6.1: Writing mechanism in Cached Uniform Writing. During non-writing intervals, the controller hidden states are pushed into the cache. When the writing time comes, the controller attends to the cache, chooses suitable states and accesses the memory. The cache is then emptied.

over several timesteps (i.e., $x_1$, ...,$x_t$). If another write happens at step $t + k$, the state $h_{t+k}$'s information containing timesteps $x_{t+1}$, ...,$x_{t+k}$ is stored in the memory ($h_{t+k}$ may involves timesteps further to the past, yet they are already stored in the previous write and can be ignored). During writing, overwriting may happen, replacing an old write with a new one. Thus after all, $D$ memory slots associate with $D$ chosen writes of the controller. From these observations, we can generalise Theorem 6.1 to the case of MANNs having $D$ memory slots (see Appendix D.5 for proof).

**Theorem 6.2.** *With any $D$ chosen writes at timesteps $1 \leq K_1 < K_2 < ... < K_D < T$, there exist $\lambda, C \in \mathbb{R}^+$ such that the lower bound on the average contribution of a sequence of length $T$ with respect to a MANN having $D$ memory slots can be quantified as the following,*

$$
\begin{aligned}
I_\lambda = C &\frac{\sum\limits_{t=1}^{K_1} \lambda^{K_1-t} + \sum\limits_{t=K_1+1}^{K_2} \lambda^{K_2-t} + ... + \sum\limits_{t=K_{D-1}+1}^{K_D} \lambda^{K_D-t} + \sum\limits_{t=K_D+1}^{T} \lambda^{T-t}}{T} \\
&= \frac{C}{T} \sum_{i=1}^{D+1} \sum_{j=0}^{l_i-1} \lambda^j = \frac{C}{T} \sum_{i=1}^{D+1} f_\lambda(l_i)
\end{aligned}
\tag{6.6}
$$

$$
\text{where } l_i = \begin{cases} K_1 & ; i = 1 \\ K_i - K_{i-1} & ; D \geq i > 1 \end{cases}, \; f_\lambda(x) = \begin{cases} \frac{1-\lambda^x}{1-\lambda} & \lambda \neq 1 \\ x & \lambda = 1 \end{cases}, \; \forall x \in \mathbb{R}^+.
$$

If $\lambda \leq 1$, we want to maximise $I_\lambda$ to keep the information from vanishing. On the contrary, if $\lambda > 1$, we may want to minimise $I_\lambda$ to prevent the information explosion. As both scenarios share the same solution (see Appendix D.6), thereafter we assume that $\lambda \leq 1$ holds for other analyses. By taking average over $T$, we are making an assumption that all timesteps are equally important. This helps simplify the measurement as $I_\lambda$ is independent of the specific position of writing. Rather, it is a function of the interval lengths between the writes. This turns out to be an optimisation problem whose solution is stated in the following theorem.

**Theorem 6.3.** *Given $D$ memory slots, a sequence with length $T$, a decay rate $0 < \lambda \leq 1$, then the optimal intervals $\{l_i \in \mathbb{R}^+\}_{i=1}^{D+1}$ satisfying $T = \sum\limits_{i=1}^{D+1} l_i$ such that the lower bound on the average contribution $I_\lambda = \frac{C}{T} \sum\limits_{i=1}^{D+1} f_\lambda(l_i)$ is maximised are*

$$
l_1 = l_2 = ... = l_{D+1} = \frac{T}{D+1} \tag{6.7}
$$

We name the optimal solution as Uniform Writing (UW) and refer to the term $\frac{T}{D+1}$ and $\frac{D+1}{T}$ as the *optimal interval* and the *compression ratio*, respectively. The proof is given in Appendix D.6.

## 6.4 Optimal Writing for Slot-based Memory Models

### 6.4.1 Uniform Writing

Uniform writing can apply to any MANNs that support writing operations. Since the writing intervals are discrete, i.e., $l_i \in \mathbb{N}^+$, UW is implemented as the following,

$$
M_t = \begin{cases} f_w(o_t, M_{t-1}) & \text{if } t = \lfloor \frac{T}{D+1} \rfloor k, k \in \mathbb{N}^+ \\ M_{t-1} & \text{otherwise} \end{cases} \tag{6.8}
$$

---

**Algorithm 6.1** Cached Uniform Writing

---

**Require:** a sequence $x = \{x_t\}_{t=1}^{T}$, a cache $C$ sized $L$, a memory sized $D$.
 1: **for** $t = 1, T$ **do**
 2:     $C$.append($h_{t-1}$)
 3:     **if** $t \bmod L == 0$ **then**
 4:         Use Eq.(6.10) to calculate $a_t$
 5:         Execute Eq.(6.3): $o_t = f_o(a_t, r_{t-1}, x_t)$
 6:         Execute Eq.(6.4): $h_t = f_h(a_t, r_{t-1}, x_t)$
 7:         Update the memory using Eq.(6.2)
 8:         Read $r_t$ from the memory using Eq.(6.1)
 9:         $C$.clear()
10:     **else**
11:         Update the controller using Eq.(6.4): $h_t = f_h(h_{t-1}, r_{t-1}, x_t)$
12:         Assign $r_t = r_{t-1}$
13:     **end if**
14: **end for**

---

By following Eq. (6.8), the write intervals are close to the optimal interval defined in Theorem 6.3 and approximately maximise the average contribution. This writing policy works well if timesteps are equally important and the task is to remember all of them to produce outputs (i.e., in copy task). However, in reality, timesteps are not created equal and a good model may need to ignore unimportant or noisy timesteps. That is why overwriting in MANN can be necessary. In the next section, we propose a method that tries to balance between following the optimal strategy and employing overwriting mechanism as in current MANNs.

## 6.4.2   Local Optimal Design

To relax the assumptions of Theorem 6.3, we propose two improvements of the Uniform Writing (UW) strategy. First, the intervals between writes are equal with length $L$ ($1 \leq L \leq \left\lfloor \frac{T}{D+1} \right\rfloor$). If $L = 1$, the strategy becomes regular writing and if $L = \left\lfloor \frac{T}{D+1} \right\rfloor$, it becomes uniform writing. This ensures that after $\left\lfloor \frac{T}{L} \right\rfloor$ writes, all memory slots should be filled and the model has to learn to overwrite. Meanwhile, the average kept information is still locally maximised every $L * D$ timesteps.

Second, we introduce a cache of size $L$ to store the hidden states of the controller during a write interval. Instead of using the hidden state at the writing timestep to update the memory, we perform an attention over the cache to choose the best representative hidden state. The model will learn to assign attention weights to the elements in the cache. This mechanism helps the model consider the impor-

tance of each timestep input in the local interval and thus relax the equal contribution assumption of Theorem 6.3. We name the writing strategy that uses the two mentioned-above improvements as Cached Uniform Writing (CUW). An illustration of the writing mechanism is depicted in Fig. 6.1.

### 6.4.3 Local Memory-Augmented Attention Unit

In this subsection, we provide details of the attention mechanism used in our CUW. To be specific, the best representative hidden state $a_t$ is computed as follows,

$$\alpha_{tj} = softmax\left(v^T \tanh\left(Wh_{t-1} + Ud_j + Vr_{t-1}\right)\right) \tag{6.9}$$

$$a_t = \sum_{j=1}^{L} \alpha_{tj} d_j \tag{6.10}$$

where $\alpha_{tj}$ is the attention score between the $t$-th writing step and the $j$-th element in the cache; $W$, $U$, $V$ and $v$ are parameters; $h$ and $r$ are the hidden state of the controller and the read-out (Eq. (6.1)), respectively; $d_j$ is the cache element and can be implemented as the controller's hidden state ($d_j = h_{t-1-L+j}$).

The vector $a_t$ will be used to replace the previous hidden state in updating the controller and memory. The whole process of performing CUW is summarised in Algo. 6.1[1].

## 6.5 Experiments and Results

### 6.5.1 An Ablation Study: Memory-Augmented Neural Networks with and without Uniform Writing

In this section, we study the impact of uniform writing on MANNs under various circumstances (different controller types, memory types and number of memory slots).

---

[1]Source code is available at `https://github.com/thaihungle/UW-DNC`
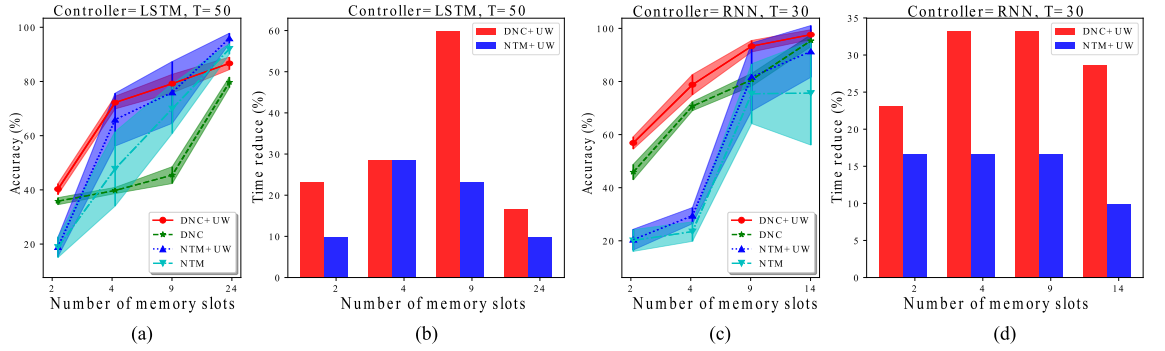
Figure 6.2: The accuracy (%) and computation time reduction (%) with different memory types and number of memory slots. The controllers/sequence lengths/memory sizes are chosen as $LSTM/50/\{2, 4, 9, 24\}$ (a&b) and $RNN/30/\{2, 4, 9, 14\}$ (c&d), respectively.

We restrict the memorisation problem to the double task in which the models must reconstruct a sequence of integers sampled uniformly from range $[1, 10]$ twice. We cast this problem to a sequence to sequence problem with 10 possible outputs per decoding step. The training stops after 10,000 iterations of batch size 64. We choose DNC[2] and NTM[3] as the two MANNs in the experiment. The recurrent controllers can be RNN or LSTM. With LSTM controller, the sequence length is set to 50. We choose sequence length of 30 to make it easier for the RNN controller to learn the task. The number of memory slots $D$ is chosen from the set $\{2, 4, 9, 24\}$ and $\{2, 4, 9, 14\}$ for LSTM and RNN controllers, respectively. More memory slots will make UW equivalent to the regular writing scheme. For this experiment, we use Adam optimiser (Kingma and Ba, 2014) with initial learning rate and gradient clipping of $\{0.001, 0.0001\}$ and $\{1, 5, 10\}$, respectively. The metric used to measure the performance is the average accuracy across decoding steps. For each configuration of hyper-parameters, we run the experiment 5 times and report the mean accuracy with error bars.

Figs. 6.2(a) and (c) depict the performance of UW and regular writing under different configurations. In any case, UW boosts the prediction accuracy of MANNs. The performance gain can be seen clearly when the compression ratio is between $10 - 40\%$. This is expected since when the compression ratio is too small or too big, UW converges to regular writing. Interestingly, increasing the memory size does not always improve the performance, as in the case of NTM with RNN controllers. Perhaps, learning to attend to many memory slots is tricky for some task given limited amount of training data. This supports the need to apply UW to MANN

---

[2]Our reimplementation based on https://github.com/deepmind/dnc
[3]https://github.com/MarkPKCollier/NeuralTuringMachine

with moderate memory size. We also conduct experiments to verify the benefit of using UW for bigger memory. The results can be found in Appendix D.8.

We also measure the speed-up of training time when applying UW on DNC and NTM, which is illustrated in Figs. 6.2(b) and (d). The result shows that with UW, the training time can drop up to 60% for DNC and 28% for NTM, respectively. As DNC is more complicated than NTM, using UW to reduce memory access demonstrates clearer speed-up in training (similar behavior can be found for testing time).

## 6.5.2   Synthetic Memorisation

Here we address a broader range of baselines on two synthetic memorisation tasks, which are the sequence copy and reverse. In these tasks, there is no discrimination amongst timesteps so the model's goal is to learn to compress the input efficiently for later retrieval. We experiment with different sequence lengths of 50 and 100 timesteps. Other details are the same as the previous double task except that we fix the learning rate and gradient clipping to 0.001 and 10, respectively. The standard baselines include LSTM, NTM and DNC. All memory-augmented models have the same memory size of 4 slots, corresponding to compression ratio of 10% and 5%, respectively. We aim at this range of compression ratio to match harsh practical requirements. UW and CUW (cache size $L = 5$) are built upon the DNC, which from our previous observations, works best for given compression ratios. We choose different dimensions $N_h$ for the hidden vector of the controllers to ensure the model sizes are approximately equivalent. To further verify that our UW is actually the optimal writing strategy, we design a new baseline, which is DNC with random irregular writing strategy (RW). The write is sampled from a binomial distribution with $p = (D + 1) / T$ (equivalent to compression ratio). After sampling, we conduct the training for that policy. The final performances of RW are taken average from 3 different random policies' results.

The performance of the models is listed in Table 6.1. As clearly seen, UW is the best performer for the pure memorisation tests. This is expected from the theory as all timesteps are importantly equivalent. Local attention mechanism in CUW does not help much in this scenario and thus CUW finishes the task as the runner-up. Reverse seems to be easier than copy as the models tend to "remember" more the last-seen timesteps whose contributions $\lambda^{T-t}$ remains significant. In both cases, other baselines including random irregular and regular writing underperform our

| Model | $N_h$ | # parameter | Copy | | Reverse | |
|---|---|---|---|---|---|---|
| | | | L=50 | L=100 | L=50 | L=100 |
| LSTM | 125 | 103,840 | 15.6 | 12.7 | 49.6 | 26,1 |
| NTM | 100 | 99,112 | 40.1 | 11.8 | 61.1 | 20.3 |
| DNC | 100 | 98,840 | 68.0 | 44.2 | 65.0 | 54.1 |
| DNC+RW | 100 | 98,840 | 47.6 | 37.0 | 70.8 | 50.1 |
| DNC+UW | 100 | 98,840 | **97.7** | **69.3** | **100** | **79.5** |
| DNC+CUW | 95 | 96,120 | 83.8 | 55.7 | 93.3 | 55.4 |

Table 6.1: Test accuracy (%) on synthetic memorisation tasks. MANNs have 4 memory slots.

proposed models by a huge margin.

### 6.5.3 Synthetic Reasoning

Tasks in the real world rarely involve just memorisation. Rather, they require the ability to selectively remember the input data and synthesise intermediate computations. To investigate whether our proposed writing schemes help the memory-augmented models handle these challenges, we conduct synthetic reasoning experiments which include add and max tasks. In these tasks, each number in the output sequence is the sum or the maximum of two numbers in the input sequence. The pairing is fixed as: $y_t = \frac{x_t + x_{T-t}}{2}, t = \overline{1, \left\lfloor \frac{T}{2} \right\rfloor}$ for add task and $y_t = \max(x_{2t}, x_{2t+1}), t = \overline{1, \left\lfloor \frac{T}{2} \right\rfloor}$ for max task, respectively. The length of the output sequence is thus half of the input sequence. A brief overview of input/output format for these tasks can be found in Appendix D.7. We deliberately use local (max) and distant (add) pairing rules to test the model under different reasoning strategies. The same experimental setting as in the previous section is applied except for the data sample range for the max task, which is $[1, 50]$[4]. LSTM and NTM are excluded from the baselines as they fail on these tasks.

Table 6.2 shows the testing results for the reasoning tasks. Since the memory size is small compared to the number of events, regular writing or random irregular writing cannot compete with the uniform-based writing policies. Amongst all baselines, CUW demonstrates superior performance in both tasks thanks to its local attention mechanism. It should be noted that the timesteps should not be treated equally in these reasoning tasks. The model should weight a timestep differently based on

---

[4]With small range like $[1, 10]$, there is no much difference in performance amongst models

| Model | Add | | Max | |
|---|---|---|---|---|
| | L=50 | L=100 | L=50 | L=100 |
| DNC | 83.8 | 22.3 | 59.5 | 27.4 |
| DNC+RW | 83.0 | 22.7 | 59.7 | 36.5 |
| DNC+UW | 84.8 | 50.9 | 71.7 | 66.2 |
| DNC+CUW | **94.4** | **60.1** | **82.3** | **70.7** |

Table 6.2: Test accuracy (%) on synthetic reasoning tasks. MANNs have 4 memory slots.

either its content (max task) or location (add task) and maintain its memory for a long time by following uniform criteria. CUW is designed to balance the two approaches and thus it achieves better performance. Further insights into memory operations of these models are given in Appendix D.9.

### 6.5.4 Synthetic Sinusoidal Regression

In real-world settings, sometimes a long sequence can be captured and fully reconstructed by memorising some of its feature points. For example, a periodic function such as sinusoid can be well-captured if we remember the peaks of the signal. By observing the peaks, we can deduce the frequency, amplitude, phase and thus fully reconstructing the function. To demonstrate that UW and CUW are useful for such scenarios, we design a sequential continuation task, in which the input is a sequence of sampling points across some sinusoid: $y = 5 + A\sin(2\pi f x + \varphi)$. Here, $A \sim \mathcal{U}(1,5)$, $f \sim \mathcal{U}(10,30)$ and $\varphi \sim \mathcal{U}(0,100)$. After reading the input $y = \{y_t\}_{t=1}^T$, the model have to generate a sequence of the following points in the sinusoid. To ensure the sequence $y$ varies and covers at least one period of the sinusoid, we set $x = \{x_t\}_{t=1}^T$ where $x_i = (t + \epsilon_1)/1000$, $\epsilon_1 \sim \mathcal{U}(-1,1)$. The sequence length for both input and output is fixed to $T = 100$. The experimental models are LSTM, DNC, UW and CUW (built upon DNC). For each model, optimal hyperparameters including learning rate and clipping size are tuned with 10,000 generated sinusoids. The memories have 4 slots and all baselines have similar parameter size. We also conduct the experiment with noisy inputs by adding a noise $\epsilon_2 \sim \mathcal{U}(-2,2)$ to the input sequence $y$. This increases the difficulty of the task. The loss is the average of mean square error (MSE) over decoding timesteps.

We plot the mean learning curves with error bars over 5 runnings for sinusoidal regression task under clean and noisy condition in Figs. 6.3(a) and (b), respectively. Regular writing DNC learns fast at the beginning, yet soon saturates and approaches
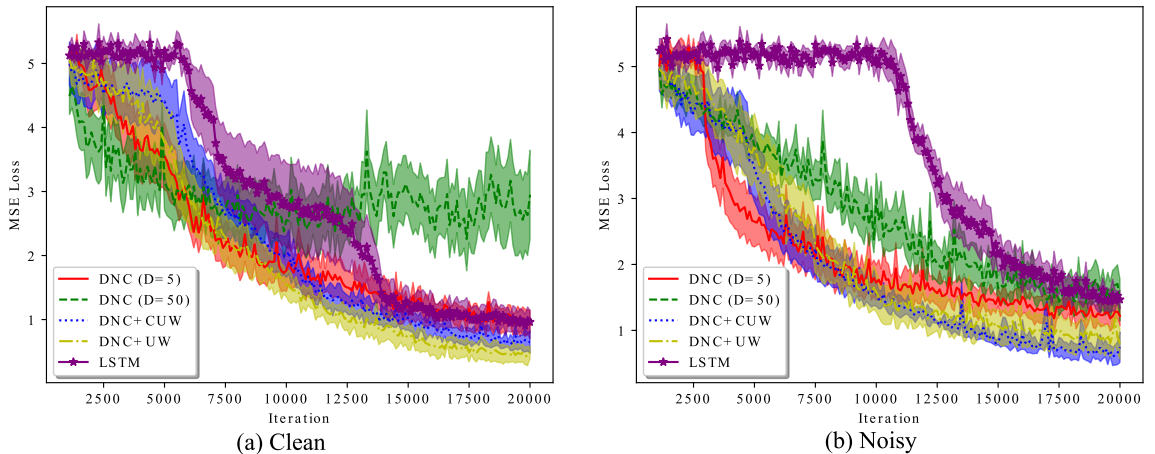
Figure 6.3: Learning curves of models in clean (a) and noisy (b) sinusoid regression experiment.

the performance of LSTM ($MSE = 1.05$ and 1.39 in clean and noisy condition, respectively). DNC performance does not improve much as we increase the memory size to 50, which implies the difficulty in learning with big memory. Although UW starts slower, it ends up with lower errors than DNC and perform slightly better than CUW in clean condition ($MSE = 0.44$ for UW and 0.61 for CUW). CUW demonstrates competitive performance against other baselines, approaching to better solution than UW for noisy task where the model should discriminate the timesteps ($MSE = 0.98$ for UW and 0.55 for CUW). More visualisations can be found in Appendix D.10.

### 6.5.5 Flatten Image Recognition

We want to compare our proposed models with DNC and other methods designed to help recurrent networks learn longer sequence. The chosen benchmark is a pixel-by-pixel image classification task on MNIST in which pixels of each image are fed into a recurrent model sequentially before a prediction is made. In this task, the sequence length is fixed to 768 with highly redundant timesteps (black pixels). The training, validation and testing sizes are 50,000, 10,000 and 10,000, respectively. We test our models on both versions of non-permutation (MNIST) and permutation (pMNIST) (Le et al., 2015). More details on the task and data can be found in Le et al., (2015). For DNC, we try with several memory slots from $\{15, 30, 60\}$ and report the best results. For UW and CUW, memory size is fixed to 15 and cache size $L$ is set to 10. The controllers are implemented as single layer GRU with 100-dimensional hidden vector. To optimise the models, we use RMSprop with initial learning rate

| Model | MNIST | pMNIST |
|---|---|---|
| iRNN[†] | 97.0 | 82.0 |
| uRNN° | 95.1 | 91.4 |
| r-LSTM Full BP[⋆] | 98.4 | 95.2 |
| Dilated-RNN[♦] | 95.5 | 96.1 |
| Dilated-GRU[♦] | **99.2** | 94.6 |
| DNC | 98.1 | 94.0 |
| DNC+UW | 98.6 | 95.6 |
| DNC+CUW | 99.1 | **96.3** |

Table 6.3: Test accuracy (%) on MNIST, pMNIST. Previously reported results are from the literature (Le et al., 2015)[†], (Arjovsky et al., 2016)°, (Trinh et al., 2018)[⋆], and (Chang et al., 2017)[♦].

of 0.0001.

Table 6.3 shows that DNC underperforms r-LSTM, which indicates that regular DNC with big memory finds it hard to beat LSTM-based methods. After applying UW, the results get better and with CUW, it shows significant improvement over r-LSTM and demonstrates competitive performance against dilated-RNNs models. Notably, dilated-RNNs use 9 layers in their experiments compared to our singer layer controller. Furthermore, our models exhibit more consistent performance than dilated-RNNs. For completeness, we include comparisons between CUW and non-recurrent methods in Appendix D.11

## 6.5.6 Document Classification

To verify our proposed models in real-world applications, we conduct experiments on document classification task. In the task, the input is a sequence of words and the output is the classification label. Following common practices (Yogatama et al., 2017; Seo et al., 2018), each word in the document is embedded into a 300-dimensional vector using Glove embedding (Pennington et al., 2014). We use RMSprop for optimisation, with initial learning rate of 0.0001. Early-stop training is applied if there is no improvement after 5 epochs in the validation set. Our UW and CUW are built upon DNC with single layer 512-dimensional LSTM controller and the memory size is chosen in accordance with the average length of the document, which ensures $10 - 20\%$ compression ratio. The cache size for CUW is fixed to 10. The datasets used in this experiment are common big datasets where the number of documents is between 120,000 and 1,400,000 with maximum of 4,392 words per

| Model | AG | IMDb[5] | Yelp P. | Yelp F. | DBP | Yah. A. |
|---|---|---|---|---|---|---|
| VDCNN[•] | 91.3 | - | 95.7 | 64.7 | 98.7 | 73.4 |
| D-LSTM[*] | - | - | 92.6 | 59.6 | 98.7 | *73.7* |
| Standard LSTM[‡] | 93.5 | 91.1 | - | - | - | - |
| Skim-LSTM[‡] | *93.6* | 91.2 | - | - | - | - |
| Region Embedding[▲] | 92.8 | - | ***96.4*** | *64.9* | *98.9* | *73.7* |
| DNC+UW | 93.7 | **91.4** | **96.4** | 65.3 | **99.0** | 74.2 |
| DNC+CUW | **93.9** | 91.3 | **96.4** | **65.6** | **99.0** | **74.3** |

Table 6.4: Document classification accuracy (%) on several datasets. Previously reported results are from the literature (Conneau et al., 2016)[•], (Yogatama et al., 2017)[*], (Seo et al., 2018)[‡] and (Qui et al., 2018)[▲]. We use italics to denote the best published and bold the best records.

document (see Appendix D.12 for further details). The baselines are recent state-of-the-arts in the domain, some of which are based on recurrent networks such as D-LSTM (Yogatama et al., 2017) and Skim-LSTM (Seo et al., 2018). We exclude DNC from the baselines as it is inefficient to train the model with big document datasets.

Our results are reported in Table 6.4. On five datasets out of six, our models beat or match the best published results. For IMDb dataset, our methods outperform the best recurrent model (Skim-LSTM). The performance gain is competitive against that of the state-of-the-arts. In most cases, CUW is better than UW, which emphasises the importance of relaxing the timestep equality assumption in practical situations. Details results across different runs for our methods are listed in Appendix D.13.

## 6.6 Closing Remarks

We have introduced Uniform Writing (UW) and Cached Uniform Writing (CUW) as faster solutions for longer-term memorisation in MANNs. With a comprehensive suite of synthetic and practical experiments, we provide strong evidences that our simple writing mechanisms are crucial to MANNs to reduce computation complexity and achieve competitive performance in sequence modeling tasks. In complement to the experimental results, we have proposed a meaningful measurement on MANN memory capacity and provided theoretical analysis showing the optimality of our

---

[5]Methods that use semi-supervised training to achieve higher accuracy are not listed.

methods. In the next chapter, we shift the focus back to designing a better MANN that is capable of learning multiple procedures at the same time and behaving like a universal Turing machine rather than a Turing machine.

# Chapter 7

# Neural Stored-Program Memory

## 7.1 Introduction

Recurrent Neural Networks (RNNs) are Turing-complete, that is, we can theoretically construct a Turing machine by using RNN elements (Siegelmann and Sontag, 1995). However, in practice RNNs struggle to learn simple procedures as they lack explicit memory (Graves et al., 2014; Mozer and Das, 1993) . Memory Augmented Neural Networks (MANNs) as analysed in Chapter 3, lift these limitations as they are capable of emulating modern computer behavior by detaching memorisation from computation via memory and controller network, respectively. In previous chapters, we have also demonstrated that MANNs are powerful in many settings, ranging from encoding multi-processes, generating complex sequences, to skip-readings. Nonetheless, MANNs have barely simulated general-purpose computers as they miss a key concept in computer design: stored-program memory.

The concept of stored-program has emerged from the idea of Universal Turing Machine (UTM) (Turing, 1936) and developed in the Von Neumann Architecture (VNA) (von Neumann, 1993). In UTM/VNA, both data and programs that manipulate the data are stored in memory. A control unit then reads the programs from the memory and executes them with the data. This mechanism allows flexibility to perform universal computations. Unfortunately, current MANNs such as Neural Turing Machine (NTM) (Graves et al., 2014), Differentiable Neural Computer (DNC) (Graves et al., 2016) and Least Recently Used Access (LRUA) (Santoro et al., 2016) only support memory for data and embed a single program into the controller

network, which goes against the stored-program memory principle.

Our goal is to advance a step further towards UTM/VNA by coupling a MANN with an external program memory. The program memory co-exists with the data memory in the MANN, providing more flexibility, reuseability and modularity in learning complicated tasks. The program memory stores the weights of the MANN's controller network, which are retrieved quickly via a key-value attention mechanism across timesteps yet updated slowly via backpropagation. By introducing a meta network to moderate the operations of the program memory, our model, henceforth referred to as Neural Stored-program Memory (NSM), can learn to switch the programs/weights in the controller network appropriately, adapting to different functionalities aligning with different parts of a sequential task, or different tasks in continual and few-shot learning.

To validate our proposal, the NTM armed with NSM, namely Neural Universal Turing Machine (NUTM), is tested on a variety of synthetic tasks including algorithmic tasks (Graves et al., 2014), composition of algorithmic tasks and continual procedure learning. For these algorithmic problems, we demonstrate clear improvements of NUTM over NTM. Further, we investigate NUTM in few-shot learning by using LRUA as the MANN and achieve notably better results. Finally, we expand NUTM application to linguistic problems by equipping NUTM with DNC core and achieve competitive performances against state-of-the-arts in the bAbI task (Weston et al., 2015).

Taken together, our study advances neural network simulation of Turing Machines to neural architecture for Universal Turing Machines. This develops a new class of MANNs that can store and query both the weights and data of their own controllers, thereby following the stored-program principle. A set of five diverse experiments demonstrate the computational universality of the approach.

## 7.2 Backgrounds

### 7.2.1 Turing Machines and MANNs

In this section, we briefly review MANN and its relations to Turing Machines. A MANN consists of a controller network and an external memory $\mathbf{M} \in \mathbb{R}^{N \times M}$, which

is a collection of $N$ $M$-dimensional vectors. The controller network is responsible for accessing the memory, updating its state and optionally producing output at each timestep. The first two functions are executed by an interface network and a state network[1], respectively. Usually, the interface network is a Feedforward neural network whose input is $c_t$ - the output of the state network implemented as RNNs. Let $W^c$ denote the weight of the interface network, then the state update and memory control are as follows,

$$h_t, c_t = RNN\left(\left[x_t, r_{t-1}\right], h_{t-1}\right) \tag{7.1}$$

$$\xi_t = c_t W^c \tag{7.2}$$

where $x_t$ and $r_{t-1}$ are data from current input and the previous memory read, respectively. The interface vector $\xi_t$ then is used to read from and write to the memory $\mathbf{M}$. We use a generic notation $memory\left(\xi_t, \mathbf{M}\right)$ to represent these memory operations that either update or retrieve read value $r_t$ from the memory. To support multiple memory accesses per step, there might be several interface networks to produce multiple interfaces, also known as control heads. Readers are referred to Graves et al., (2014, 2016) and Santoro et al., (2016) for details of memory read/write examples.

A deterministic one-tape Turing Machine can be defined by 4-tuple $(Q, \Gamma, \delta, q_0)$, in which $Q$ is finite set of states, $q_0 \in Q$ is an initial state, $\Gamma$ is finite set of symbol stored in the tape (the data) and $\delta$ is the transition function (the program), $\delta : Q \times \Gamma \to \Gamma \times \{-1, 1\} \times Q$. At each step, the machine performs the transition function, which takes the current state and the read value from the tape as inputs and outputs actions including writing new values, moving tape head to new location (left/right) and jumping to another state. Roughly mapping to current MANNs, $Q$, $\Gamma$ and $\delta$ map to the set of the controller states, the read values and the controller network, respectively. Further, the function $\delta$ can be factorised into two sub functions: $Q \times \Gamma \to \Gamma \times \{-1, 1\}$ and $Q \times \Gamma \to Q$, which correspond to the interface and state networks, respectively.

By encoding a Turing Machine into the tape, one can build an UTM that simulates the encoded machine (Turing, 1936). The transition function of the UTM queries the encoded Turing Machine that solves the considering task. Amongst 4 tuples, $\delta$ is

---

[1]Some MANNs (e.g., NTM with Feedforward Controller) neglect the state network, only implementing the interface network and thus analogous to one-state Turing Machine.

the most important and hence uses most of the encoding bits. In other words, if we assume that the space of $Q$, $\Gamma$ and $q_0$ are shared amongst Turing Machines, we can simulate any Turing Machine by encoding only its transition function $\delta$. Translating to neural language, if we can store the controller network into a queriable memory and make use of it, we can build a Neural Universal Turing Machine. Using NSM is a simple way to achieve this goal, which we introduce in the subsequent section.

## 7.2.2   Related Approaches

Previous investigations into MANNs mostly revolve around memory access mechanisms. The works in Graves et al., (2014, 2016) introduce content-based, location-based and dynamic memory reading/writing. Further, Rae et al., (2016) scales to bigger memory by sparse access while Le et al., (2019a) optimises memory operations with uniform writing. These works keep using memory for storing data rather than the weights of the network and thus parallel to our approach. Other DNC modifications (Csordas and Schmidhuber, 2019; Franke et al., 2018) are also orthogonal to our work.

Another line of related work involves modularisation of neural networks, which is designed for visual question answering. In module networks (Andreas et al., 2016b,a), the modules are manually aligned with predefined concepts and the order of execution is decided by the question. Although the module in these works resembles the program in NSM, our model is more generic and flexible with soft-attention over programs and thus fully differentiable. Further, the motivation of NSM does not limit to a specific application. Rather, NSM aims to help MANN reach general-purpose computability.

Finally, if we view NSM network as a dynamic weight generator, the program in NSM can be linked to fast weight (Hinton and Plaut, 1987; Ba et al., 2016; Munkhdalai and Yu, 2017). These papers share the idea of using different weights across timesteps to enable dynamic adaptation. However, fast weights are directly generated while our programs are interpolated from a set of slow weights.

# 7.3 Neural Stored-Program Memory and Neural Universal Turing Machine

## 7.3.1 Neural Stored-Program Memory

A Neural Stored-program Memory (NSM) is a key-value memory $\mathbf{M}_p \in \mathbb{R}^{P \times (K+S)}$, whose value is the weight of another neural network−the program. $P$, $K$, and $S$ are the number of programs, the key space dimension and the program size, respectively. This concept is a hybrid between the traditional slow-weight and fast-weight (Hinton and Plaut, 1987). Like slow-weight, the weights in NSM are updated gradually by backpropagation. However, they are dynamically recomputed on-the-fly during the processing of a sequence, which resembles fast-weight computation. Let us denote $\mathbf{M}_p(i).k$ and $\mathbf{M}_p(i).v$ as the key and the content of the $i$-th memory slot. At timestep $t$, given a query key $k_t^p$, the corresponding program is retrieved as follows,

$$D\left(k_t^p, \mathbf{M}_p(i).k\right) = \frac{k_t^p \cdot \mathbf{M}_p(i).k}{||k_t^p|| \cdot ||\mathbf{M}_p(i).k)||} \tag{7.3}$$

$$p_t = \sum_{i=1}^{P} \text{softmax}\left(\beta_t^p D\left(k_t^p, \mathbf{M}_p(i).k\right)\right) \mathbf{M}_p(i).v \tag{7.4}$$

where $D(\cdot)$ is cosine similarity and $\beta_t^p$ is the scalar program strength parameter. The vector program $p_t$ is then reshaped to its matrix form and ready to be used in other neural computations.

The key-value design is essential for convenient memory access as the size of the program stored in $\mathbf{M}_p$ can be millions of dimensions and thus, direct content-based addressing (Graves et al., 2014, 2016) is infeasible. More importantly, we can inject external control on the behavior of the memory by imposing constraints on the key space. For example, program collapse will happen when the keys stored in the memory stay close to each other. When this happens, $p_t$ is a balanced mixture of all programs regardless of the query key and thus having multiple programs is useless. We can avoid this phenomenon by minimising a regularisation loss defined as the following,

$$l_p = \sum_{i=1}^{P} \sum_{j=i+1}^{P} D\left(\mathbf{M}_p(i).k, \mathbf{M}_p(j).k\right) \tag{7.5}$$

## 7.3.2 Neural Universal Turing Machine

It turns out that the combination of MANN and NSM approximates an Universal Turing Machine (Sec. 7.2). At each timestep, the controller in MANN reads its state and memory to generate control signal to the memory via the interface network $W^c$, then updates its state using the state network $RNN$. Since the parameters of $RNN$ and $W^c$ represent the encoding of $\delta$, we store both into NSM to completely encode an MANN. For simplicity, in this proposal, we only use NSM to store $W^c$, which is equivalent to the Universal Turing Machine that can simulate any one-state Turing Machine.

In traditional MANN, $W^c$ is constant across timesteps and only updated slowly during training, typically through backpropagation. In our design, we compute $W^c_t$ from NSM for every timestep and thus, we need a program interface network−the meta network $P_{\mathcal{I}}$−that generates an interface vector for the program memory: $\xi^p_t = P_{\mathcal{I}}(c_t)$, where $\xi^p_t = [k^p_t, \beta^p_t]$ and $P_{\mathcal{I}}$ is implemented as a Feedforward neural network. The procedure for computing $W^c_t$ is executed by following Eqs. (7.3)-(7.4), hereafter referred to as $NSM\left(\xi^p_t, \mathbf{M}_p\right)$. Figure 7.1 depicts the integration of NSM into MANN.

For the case of multi-head NTM, we implement one NSM per control head and name this model Neural Universal Turing Machine (NUTM). Each control head will read from (for read head) or write to (for write head) the data memory $\mathbf{M}$ via $memory\left(\xi_t, \mathbf{M}\right)$ as described in Graves et al., (2014). Other MANNs such as DNC (Graves et al., 2016) and LRUA (Santoro et al., 2016) can be armed with NSM in this manner. We also employ the regularisation loss $l_p$ to prevent the programs from collapsing, resulting in a final loss as follows,

$$Loss = Loss_{pred} + \eta_t l_p \tag{7.6}$$

where $Loss_{pred}$ is the prediction loss and $\eta_t$ is annealing factor, reducing as the training step increases. The details of NUTM operations are presented in Algorithm 7.1.
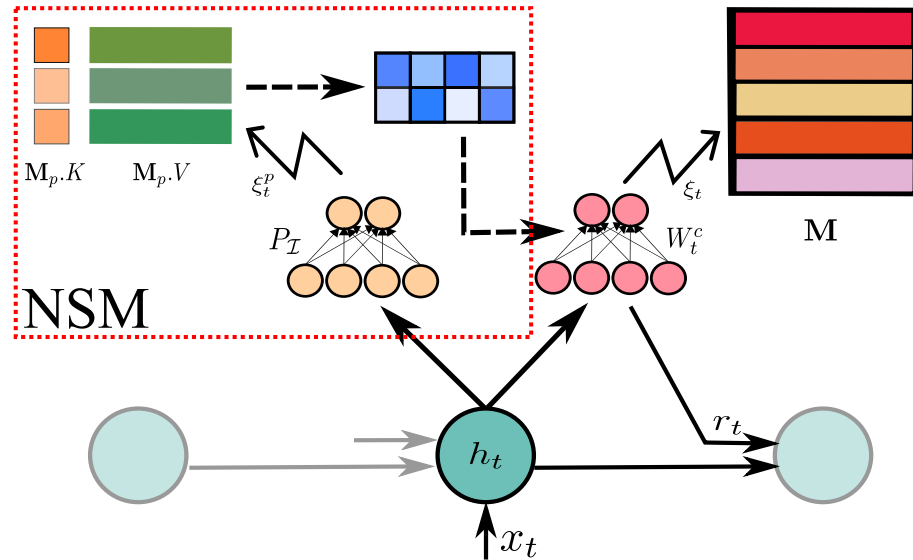
Figure 7.1: Introducing NSM into MANN. At each timestep, the program interface network ($P_{\mathcal{I}}$) receives input from the state network and queries the program memory $\mathbf{M}_p$, acquiring the working weight for the interface network ($W_t^c$). The interface network then operates on the data memory $\mathbf{M}$.

## 7.3.3 On the Benefit of NSM to MANN: An Explanation from Multilevel Modeling

Learning to access memory is a multi-dimensional regression problem. Given the input $c_t$, which is derived from the state $h_t$ of the controller, the aim is to generate a correct interface vector $\xi_t$ via optimising the interface network. Instead of searching for one transformation that maps the whole space of $c_t$ to the optimal space of $\xi_t$, NSM first partitions the space of $c_t$ into subspaces, then finds multiple transformations, each of which covers subspace of $c_t$. The program interface network $P_{\mathcal{I}}$ is a meta learner that routes $c_t$ to the appropriate transformation, which then maps $c_t$ to the $\xi_t$ space. This is analogous to multilevel regression in statistics (Andrew Gelman, 2006). Many practical studies have demonstrated that multilevel regression is better than ordinary regression if the input is clustered (Cohen et al., 2014; Huang, 2018).

RNNs have the capacity to learn to perform finite state computations (Casey, 1996; Tiňo et al., 1998). The states of a RNN must be grouped into partitions representing the states of the generating automation. As Turing Machine is finite state automata augmented with an external memory tape, we expect MANN, if learnt well, will organise its state space clustered in a way to reflect the states of the emulated Turing Machine. That is, $h_t$ as well as $c_t$ should be clustered. We realise that NSM

---

**Algorithm 7.1** Neural Universal Turing Machine

---

**Require:** a sequence $x = \{x_t\}_{t=1}^T$, a data memory $\mathbf{M}$ and $R$ program memories $\{\mathbf{M}_{p,n}\}_{n=1}^R$ corresponding to $R$ control heads
1: Initialise $h_0$, $r_0$
2: **for** $t = 1, T$ **do**
3:     $h_t, c_t = RNN([x_t, r_{t-1}], h_{t-1})$       $\triangleright$ $RNN$ can be replaced by GRU/LSTM
4:     **for** $n = 1, R$ **do**
5:         Compute the program interface $\xi_{t,n}^p \leftarrow P_{\mathcal{I},n}(c_t)$
6:         Compute the program $W_{t,n}^c \leftarrow NSM\left(\xi_{t,n}^p, \mathbf{M}_{p,n}\right)$
7:         Compute the data interface $\xi_{t,n} \leftarrow c_t W_{t,n}^c$
8:         Access/update data memory $r_{t,n} \leftarrow memory(\xi_{t,n}, \mathbf{M})$     $\triangleright$ Write heads
    return $\emptyset$
9:     **end for**
10:     $r_t \leftarrow [r_{t,1}, ..., r_{t,R}]$
11: **end for**

---
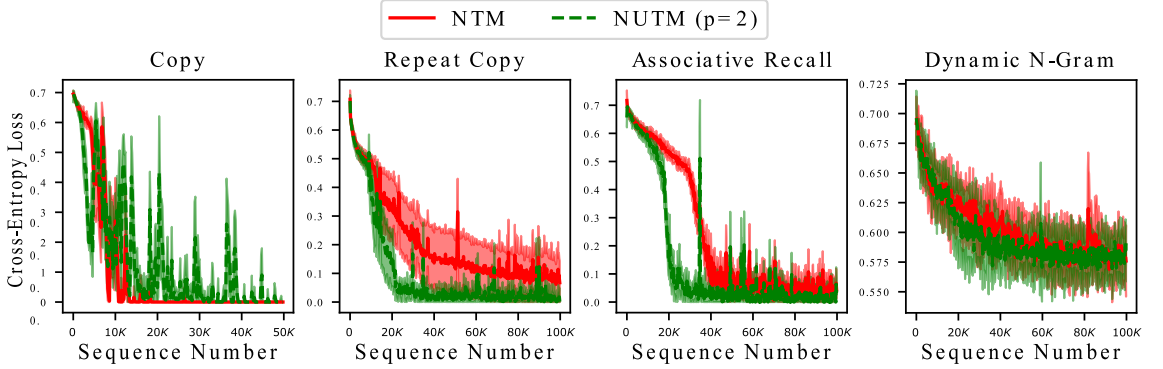


Figure 7.2: Learning curves on NTM tasks.

helps NTM learn better clusterisation over this space (see Appendix E.2), thereby improving NTM's performances.

## 7.4 Applications

### 7.4.1 NTM Single Tasks

In this section, we investigate the performance of NUTM on algorithmic tasks introduced in Graves et al., (2014): Copy, Repeat Copy, Associative Recall, Dynamic N-Gram and Priority Sort. Besides these five NTM tasks, we add another task named Long Copy which doubles the length of training sequences in the Copy task. In these tasks, the model will be fed a sequence of input items and is required to

| Task | Copy | Repeat Copy | A. Recall | D. N-grams | Priority Sort | Long Copy |
|------|------|-------------|-----------|------------|---------------|-----------|
| NTM | **0.00** | 405.10 | 7.66 | 132.59 | 24.41 | 16.04 |
| NUTM (p=2) | **0.00** | **366.69** | **1.35** | **127.68** | **20.00** | **0.02** |

Table 7.1: Generalisation performance of best models measured in average bit error per sequence (lower is better). For each task, we pick a set of 1,000 unseen sequences as test data.

infer a sequence of output items. Each item is represented by a binary vector.

In the experiment, we compare two models: NTM[2] and NUTM with two programs. Although the tasks are atomic, we argue that there should be at least two memory manipulation schemes across timesteps, one for encoding the inputs to the memory and another for decoding the output from the memory. The two models are trained with cross-entropy objective function under the same setting as in Graves et al., (2014). For fair comparison, the controller hidden dimension of NUTM is set smaller to make the total number of parameters of NUTM equivalent to that of NTM (details in Appendix E.4).

We run each experiments five times and report the mean with error bars of training losses for the first 4 tasks in Fig. E.1. Except for the Copy task, which is too simple, other tasks observe convergence speed improvement of NUTM over that of NTM, thereby validating the benefit of using two programs across timesteps even for the single task setting. Full report is listed in Appendix E.1. As NUTM requires fewer training samples to converge, it generalises better to unseen sequences that are longer than training sequences. Table 7.1 reports the test results of the best models chosen after five runs and confirms the outperformance of NUTM over NTM for generalisation.

To illustrate the program usage, we plot NUTM's program distributions across timesteps for Repeat Copy and Priority Sort in Fig. 7.3 (a) and (b), respectively. We observe two program usage patterns corresponding to the encoding and decoding phases. For Repeat Copy, there is no reading in encoding and thus, NUTM assigns the "no-read" strategy mainly to the "orange program". In decoding, the sequential reading is mostly done by the "blue program" with some contributions from the "orange program" when resetting reading head. For Priority Sort, while the encoding "fitting writing" (see Graves et al., (2014) for explanation on the strategy) is often

---

[2]For algorithmic tasks, we choose NTM as the only baseline as NTM is known to perform and generalise well on these tasks. If NSM can help NTM in these tasks, it will probably help other MANNs as well.
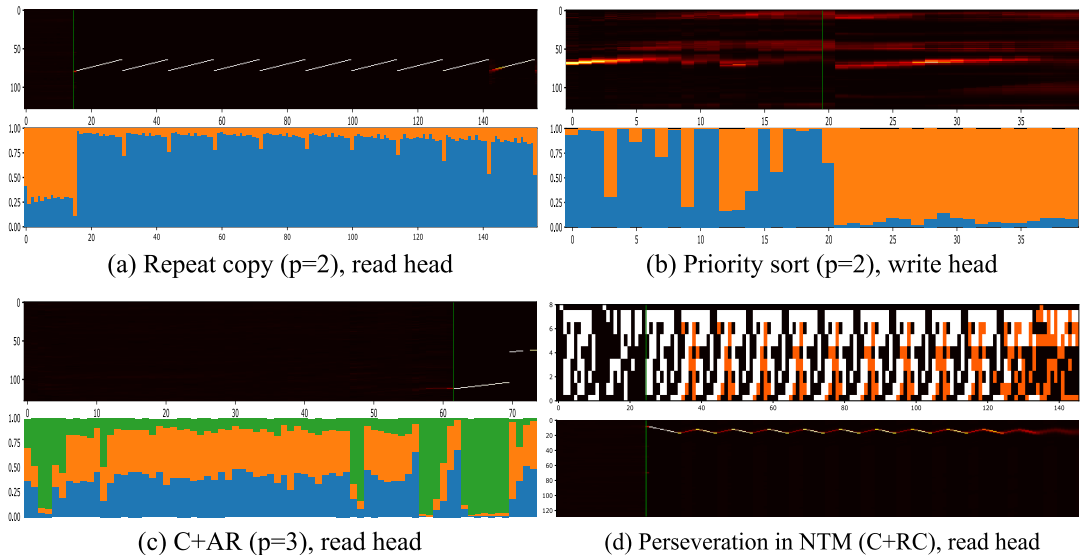
(a) Repeat copy (p=2), read head



(b) Priority sort (p=2), write head



(c) C+AR (p=3), read head



(d) Perseveration in NTM (C+RC), read head

Figure 7.3: (a,b,c) visualises NUTM's executions in synthetic tasks: the upper rows are memory read (left)/write (right) locations; the lower rows are program distributions over timesteps. The green line indicates the start of the decoding phase. (d) visualises perservation in NTM: the upper row are input, output, predicted output with errors (orange bits); the lower row is reading location.

executed by the "blue program", the decoding writing is completely taken by the "orange" program (more visualisations in Appendix E.3).

## 7.4.2 NTM Sequencing Tasks

In neuroscience, sequencing tasks test the ability to remember a series of tasks and switch tasks alternatively (Blumenfeld, 2010). A dysfunctional brain may have difficulty in changing from one task to the next and get stuck in its preferred task (perseveration phenomenon). To analyse this problem in neural algorithmic learners, we propose a new set of experiments in which a task is generated by sequencing a list of subtasks. The set of subtasks is chosen from the NTM single tasks (excluding Dynamic N-grams for format discrepancy) and the order of subtasks in the sequence is dictated by an indicator vector put at the beginning of the sequence. Amongst possible combinations of subtasks, we choose {Copy, Repeat Copy}(C+RC), {Copy, Associative Recall} (C+AR), {Copy, Priority Sort} (C+PS) and all (C+RC+AC+PS)[3]. The learner observes the order indicator following by a sequence of subtasks' input items and is requested to consecutively produce the output items of each subtasks.

---

[3]We focus on the combinations that contain Copy as Copy is the only task where NTM can reach NUTM's performance. If NTM fails in these combinations, it will most likely fail in other combinations.
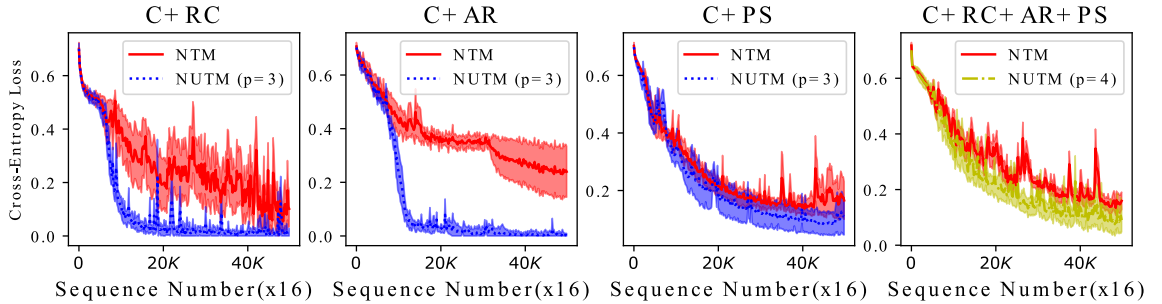
Figure 7.4: Learning curves on sequencing NTM tasks.

As shown in Fig. 7.4, some tasks such as Copy and Associative Recall, easy to solve if trained separately, become unsolvable by NTM when sequenced together. One reason for NTM's poor performance is its failure to change the memory access behavior (perseveration). For example, NTM keeps following repeat copy reading strategy for all timesteps in C+RC task (Fig. 7.3 (d)). Meanwhile, NUTM can learn to change program distribution when a new subtask appears in the sequence and thus ensure different memory accessing strategy per subtask (Fig. 7.3 (c)).

### 7.4.3 Continual Procedure Learning

In continual learning, catastrophic forgetting happens when a neural network quickly forgets previously acquired skills upon learning new skills (French, 1999). In this section, we prove the versatility of NSM by showing that a naive application of NSM without much modification can help NTM to mitigate catastrophic forgetting. We design an experiment similar to the Split MNIST (Zenke et al., 2017) to investigate whether NSM can improve NTM's performance. In our experiment, we let the models see the training data from the 4 tasks: Copy (C), Repeat Copy (RC), Associative Recall (AR) and Priority Sort (PS), consecutively in this order. Each task is trained in 20,000 iterations with batch size 16 (see Appendix E.4 for task details). To encourage NUTM to spend exactly one program per task while freezing others, we force "hard" attention over the programs by replacing the softmax function in Eq. 7.4 with the Gumbel-softmax (Jang et al., 2016). Also, to ignore catastrophic forgetting in the state network, we use Feedforward controllers in the two baselines.

After finishing one task, we evaluate the bit accuracy −measured by 1−(bit error per sequence/total bits per sequence)−over 4 tasks. As shown in Fig. 7.5, NUTM outperforms NTM by a moderate margin (10-40% per task). Although NUTM also experiences catastrophic forgetting, it somehow preserves some memories of
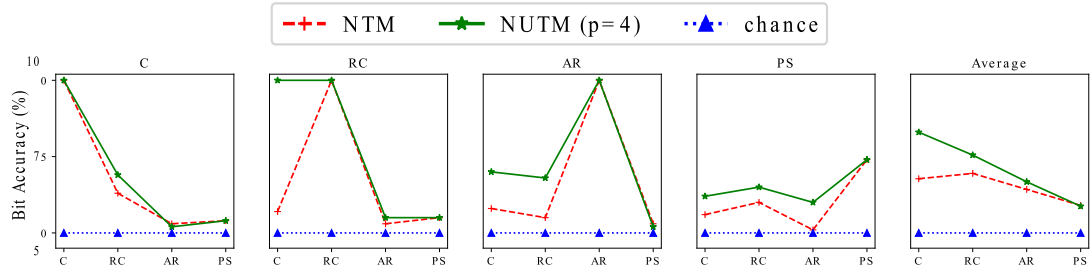
Figure 7.5: Mean bit accuracy for the continual algorithmic tasks. Each of the first four panels show bit accuracy on four tasks after finishing a task. The rightmost shows the average accuracy.

previous tasks. Especially, NUTM keeps performing perfectly on Copy even after it learns Repeat Copy. For other dissimilar task transitions, the performance drops significantly, which requires more effort to bring NSM to continual learning.

### 7.4.4 Few-Shot Learning

Few-shot learning or meta learning tests the ability to rapidly adapt within a task while gradually capturing the way the task structure varies (Thrun, 1998). By storing sample-class bindings, MANNs are capable of classifying new data after seeing only few samples (Santoro et al., 2016). As NSM gives flexible memory controls, it makes MANN more adaptive to changes and thus perform better in this setting. To verify that, we apply NSM to the LRUA memory and follow the experiments introduced in Santoro et al., (2016), using the Omniglot dataset to measure few-shot classification accuracy. The dataset includes images of 1623 characters, with 20 examples of each character. During training, a sequence (episode) of images are randomly selected from $C$ classes of characters in the training set (1200 characters), where $C = 5, 10$ corresponding to sequence length of 50, 75, respectively. Each class is assigned a random label which shuffles between episodes and is revealed to the models after each prediction. After 100,000 episodes of training, the models are tested with unseen images from the testing set (423 characters). The two baselines are MANN and NUTM (both use LRUA core). For NUTM, we only tune $p$ and pick the best values: $p = 2$ and $p = 3$ for 5 classes and 10 classes, respectively.

Table 7.2 reports the classification accuracy when the models see characters for the second, third and fifth time. NUTM generally achieves better results than MANN, especially when the number of classes increases, demanding more adaptation within an episode. For the persistent memory mode, which demands fast forgetting old

| Model | Persistent memory[4] | 5 classes | | | 10 classes | | |
|---|---|---|---|---|---|---|---|
| | | $2^{nd}$ | $3^{rd}$ | $5^{th}$ | $2^{nd}$ | $3^{rd}$ | $5^{th}$ |
| MANN (LRUA)* | No | 82.8 | 91.0 | 94.9 | - | - | - |
| MANN (LRUA) | No | 82.3 | 88.7 | 92.3 | 52.7 | 60.6 | 64.7 |
| NUTM (LRUA) | No | **85.7** | **91.3** | **95.5** | **68.0** | **78.1** | **82.8** |
| MANN (LRUA) | Yes | 66.2 | 73.4 | 81.0 | 51.3 | 59.2 | 63.3 |
| NUTM (LRUA) | Yes | **77.8** | **85.8** | **89.8** | **69.0** | **77.9** | **82.7** |

Table 7.2: Test-set classification accuracy (%) on the Omniglot dataset after 100,000 episodes of training. * denotes available results from Santoro et al., (2016). See Appendix E.5 for more details.

experiences in previous episodes, NUTM outperforms MANN significantly (10-20%).

## 7.4.5   Text Question Answering

Reading comprehension typically involves an iterative process of multiple actions such as reading the story, reading the question, outputting the answers and other implicit reasoning steps (Weston et al., 2015). We apply NUTM to the question answering domain by replacing the NTM core with DNC (Graves et al., 2016). Compared to NTM's sequential addressing, dynamic memory addressing in DNC is more powerful and thus suitable for NSM integration to solve non-algorithmic problems such as question answering. Following previous works of DNC, we use bAbI dataset (Weston et al., 2015) to measure the performance of the NUTM with DNC core (two variants $p = 2$ and $p = 4$). In the dataset, each story is followed by a series of questions and the network reads all word by word, then predicts the answers. Although synthetically generated, bAbI is a good benchmark that tests 20 aspects of natural language reasoning including complex skills such as induction, counting and path finding,

We found that NUTM with 4 programs, after 50 epochs jointly trained on all 20 question types, can achieve a mean test error rate of 3.3% and manages to solve 19/20 tasks (a task is considered solved if its error <5%). The mean and s.d. across 10 runs are also compared with other results reported by recent works (see Table 7.3). Excluding baselines under different setups, our result is the best reported mean result on bAbI that we are aware of. More details are described in Appendix E.6.

---

[4]If the memory is not artificially erased between episodes, it is called persistent. This mode is hard for the case of 5 classes (Santoro et al., 2016)

| Model | Error |
|---|---|
| DNC(Graves et al., 2016) | $16.7 \pm 7.6$ |
| SDNC(Rae et al., 2016) | $6.4 \pm 2.5$ |
| ADNC(Franke et al., 2018) | $6.3 \pm 2.7$ |
| DNC-MD(Csordas and Schmidhuber, 2019) | $9.5 \pm 1.6$ |
| NUTM (DNC core) $p = 2$ | $7.5 \pm 1.6$ |
| NUTM (DNC core) $p = 4$ | $\mathbf{5.6 \pm 1.9}$ |

Table 7.3: Mean and s.d. for bAbI error (%).

## 7.5 Closing Remarks

This chapter introduces the Neural Stored-program Memory (NSM), a new type of external memory for neural networks. The memory, which takes inspirations from the stored-program memory in computer architecture, gives memory-augmented neural networks (MANNs) flexibility to change their control programs through time while maintaining differentiability. The mechanism simulates modern computer behavior, potential making MANNs truly neural computers. Our experiments demonstrated that when coupled with our model, the Neural Turing Machine learns algorithms better and adapts faster to new tasks at both sequence and sample levels. When used in few-shot learning, our method helps MANN as well. We also applied the NSM to the Differentiable Neural Computer and observed a significant improvement, reaching the state-of-the-arts in the bAbI task.

# Chapter 8

# Conclusions

## 8.1 Summary

In this thesis, we have presented several types of memory for neural networks in general and Recurrent Neural Networks (RNNs) in particular. We emphasise the notion of the memory as an external storage for RNNs in which, RNNs can learn to read from and write to the external memory to support their working memory (Chapter 2). We reviewed advancements to solve the difficulties in training RNNs such as gating and attention mechanisms and especially we focused on slot-based MANNs, which is based on sparse distributed memory model–the main theme of new models we propose in the thesis (Chapter 3). Our main contributions are four-fold. First, we extended MANNs as a multi-process multi-view model to handle complex problems such as sequence-to-sequence mapping and multi-view sequential learning (Chapter 4). We further extended MANNs as a model for discrete sequence generation on conversational data where variability and coherence are required (Chapter 5). We also shed light into memory operations to estimate MANN capacity and proposed new writing schemes to maximise the capacity (Chapter 6). Finally, we introduced a new class of MANNs that follows stored-program memory principle and can switch the controller's programs to execute different functionalities through time.

In Chapter 4, we presented our first contributions: Dual Controller Write-Protected Memory Augmented Neural Network (DCw-MANN), an extension of MANN to model sequence to sequence mapping, and Dual Memory Neural Computer (DMNC) that can capture the correlations between two views by exploiting two external

memory units. We demonstrated these models on predicting next disease stages and recommending medicines in healthcare. The results are competitive against contemporary state-of-the-arts.

Chapter 5 presented Variational Memory Encoder-Decoder, a novel memory-augmented generation framework. Our external memory not only holds long-term context but also constructs mixture model distribution generating the latent variables. We derived theoretical analysis to validate our training protocol using Stochastic Gradient Variational Bayes framework by minimising variational approximation of KL divergence. We evaluated our model on two open-domain and two closed-domain conversational datasets and outperformed other baselines by a large margin.

Chapter 6 focused more on theoretical analysis of a meaningful measurement on MANN's memory capacity. We proposed solution dubbed Uniform Writing is optimal in terms of maximising the memory capacity. To encourage forgetting when necessary, we introduce modifications to the original solution, resulting in a new solution termed Cached Uniform Writing. This method aims to balance between memorising and forgetting via allowing overwriting mechanism. We conducted a set of diverse experiments on six ultra-long sequential learning problems given a limited number of memory slots to demonstrate the superiority of our methods.

Chapter 7 addressed the simulation capacity of MANNs. To make current MANNs truly simulate modern computers, a design of Neural Stored-program Memory (NSM) was proposed to implement stored-program principle, which resulted in new MANN architectures that materialise Universal Turing Machines.

## 8.2   Future Directions

There are possible extensions of the work proposed in this thesis for further investigations. First, in Chapter 4, we can extend the DCw-MANN to handle multiple healthcare tasks by developing new capabilities for medical question answering, which can be cast to a sequence-to-sequence mapping problem. Moreover, DMNC can be generalised to multi-input multi-output settings and extending the range of applications to bigger problems such as multi-media and multi-agent systems. For VMED (Chapter 5), future explorations may involve implementing a dynamic number of modes that enable learning of the optimal $K$ for each timestep. Another aspect would be multi-person dialog setting, where our memory as mixture model

may be useful to capture more complex modes of speaking in the dialog. Future investigations for Chapter 6 will focus on tightening the measurement bound. Last but not least, we emphasise that apart from MANNs, other neural networks can also reap benefits from the NSM proposed in Chapter 7. More effort will be spent on integrating NSM into different neural networks to make them more flexible and modular.

# Appendix

## C Supplementary for Chapter 5

### C.1 Proof of Theorem 5.1

*Proof.* $D_{var}(f \parallel g)$ (Hershey and Olsen, 2007) is an approximation of $KL$ divergence between two Mixture of Gaussians (MoG), which is defined as the following,

$$D_{var}\left(f \parallel g\right) = \sum_j \pi_j^f \log \frac{\sum_{j'} \pi_{j'}^f e^{-KL\left(f_j \| f_{j'}\right)}}{\sum_i \pi_i^g e^{-KL\left(f_j \| g_i\right)}} \tag{8.1}$$

In our case, $f$ is a Gaussian, a special case of MoG where the number of mode equals one. Then, Eq. (8.1) becomes

$$D_{var}\left(f \parallel g\right) = \log \frac{1}{\sum_{i=1}^{K} \pi_i^g e^{-KL(f\|g^i)}} = -\log \sum_{i=1}^{K} \pi^i e^{-KL\left(f\|g^i\right)}$$

Let define the log-likelihood $L_f\left(g\right) = E_{f(x)}\left[\log g\left(x\right)\right]$, the lower bound for $L_f\left(g\right)$ can be also be derived, using variational parameters as follows,

$$L_f(g) = E_f\left[\log\left(\sum_{i=1}^{K}\pi^i g^i(x)\right)\right]$$

$$= \int_{-\infty}^{+\infty} f(x)\log\left(\sum_{i=1}^{K}\beta^i\pi^i\frac{g^i(x)}{\beta^i}\right)dx$$

$$\geq \sum_{i=1}^{K}\beta^i\int_{-\infty}^{+\infty} f(x)\log\left(\pi^i\frac{g^i(x)}{\beta^i}\right)dx$$

where $\beta^i \geq 0$ and $\sum_{i=1}^{K}\beta^i = 1$. According to Durrieu et al., (2012), maximising the RHS of the above inequality with respect to $\beta^i$ provides a lower bound for $L_f(g)$ (Durrieu et al., 2012),

$$L_f(g) \geq \log\sum_{i=1}^{K}\pi^i e^{-KL\left(f\|g^i\right)} + L_f(f)$$

$$= -D_{var} + L_f(f)$$

$$\Rightarrow D_{var} \geq L_f(f) - L_f(g)$$

$$= KL(f \parallel g)$$

Therefore, the $KL$ divergence has an upper bound: $D_{var}$. □

## C.2   Derivation of the Upper Bound on the Total Timestep-Wise $KL$ Divergence

**Lemma C.1.** *Chebyshev's sum inequality:*
*if*

$$a_1 \geq a_2 \geq ... \geq a_n$$

*and*

$$b_1 \geq b_2 \geq ... \geq b_n$$

then

$$\frac{1}{n}\sum_{k=1}^{n}a_k b_k \geq \left(\frac{1}{n}\sum_{k=1}^{n}a_k\right)\left(\frac{1}{n}\sum_{k=1}^{n}b_k\right)$$

*Proof.* Consider the sum

$$S = \sum_{j=1}^{n}\sum_{k=1}^{n}(a_j - a_k)(b_j - b_k)$$

The two sequences are non-increasing, therefore $a_j - a_k$ and $b_j - b_k$ have the same sign for any $j, k$. Hence $S \geq 0$. Opening the brackets, we deduce

$$0 \leq 2n\sum_{j=1}^{n}a_j b_j - 2\sum_{j=1}^{n}a_j \sum_{k=1}^{n}b_k$$

whence

$$\frac{1}{n}\sum_{j=1}^{n}a_j b_j \geq \left(\frac{1}{n}\sum_{j=1}^{n}a_j\right)\left(\frac{1}{n}\sum_{k=1}^{n}b_k\right)$$

$\square$

In our problem, $a_i = f_i(x)$ and $b_i = \log[g_i(x)]$, $i = \overline{1, T}$. Under the assumption that at each step, thanks to minimising $D_{var}$, the approximation between the MoG and the Gaussian is adequate to preserve the order of these values, that is, if $f_i(x) \leq f_j(x)$, then $g_i(x) \leq g_j(x)$ and $\log[g_i(x)] \leq \log[g_j(x)]$. Without loss of generality, we hypothesise that $f_1(x) \leq f_2(x) \leq ... \leq f_T(x)$, then we have $\log[g_1(x)] \leq \log[g_2(x)] \leq ... \leq \log[g_T(x)]$. Thus, applying Lemma C.1, we have

$$\frac{1}{T}\sum_{t=1}^{T}f_t(x)\log[g_t(x)]\,dx \geq \frac{1}{T}\sum_{t=1}^{T}f_t(x)\frac{1}{T}\sum_{t=1}^{T}\log[g_t(x)]\,dx$$

$$\Rightarrow \int_{-\infty}^{+\infty}\sum_{t=1}^{T}f_t(x)\log[g_t(x)]\,dx \geq \int_{-\infty}^{+\infty}\frac{1}{T}\sum_{t=1}^{T}f_t(x)\sum_{t=1}^{T}\log[g_t(x)]\,dx$$

$$\Rightarrow \int_{-\infty}^{+\infty}\sum_{t=1}^{T}f_t(x)\log[g_t(x)]\,dx \geq \int_{-\infty}^{+\infty}\frac{1}{T}\sum_{t=1}^{T}f_t(x)\log\left[\prod_{t=1}^{T}g_t(x)\right]dx$$

Thus, the upper bound on the total timestep-wise $KL$ divergence reads

$$\int\limits_{-\infty}^{+\infty} \sum_{t=1}^{T} f_t(x) \log\left[f_t(x)\right] dx \quad - \int\limits_{-\infty}^{+\infty} \frac{1}{T} \sum_{t=1}^{T} f_t(x) \log\left[\prod_{t=1}^{T} g_t(x)\right] dx$$

## C.3 Proof $\prod\limits_{t=1}^{T} g_t(x) = \prod\limits_{t=1}^{T} \sum\limits_{i=1}^{K} \pi_t^i g_t^i(x)$ Is a Scaled MoG

**Lemma C.2.** *Product of two Gaussians is a scaled Gaussian.*

*Proof.* Let $\mathcal{N}_x(\mu, \Sigma)$ denote a density of $x$, then

$$\mathcal{N}_x(\mu_1, \Sigma_1) \cdot \mathcal{N}_x(\mu_2, \Sigma_2) = c_c \mathcal{N}_x(\mu_c, \Sigma_c)$$

where

$$c_c = \frac{1}{\sqrt{\det(2\pi(\Sigma_1 + \Sigma_2))}} \exp\left(-\frac{1}{2}(m_1 - m_2)^T (\Sigma_1 + \Sigma_2)^{-1} (m_1 - m_2)\right)$$
$$m_c = \left(\Sigma_1^{-1} + \Sigma_2^{-1}\right)^{-1} \left(\Sigma_1^{-1} m_1 + \Sigma_2^{-1} m_2\right)$$
$$\Sigma_c = \left(\Sigma_1^{-1} + \Sigma_2^{-1}\right)$$

$\square$

**Lemma C.3.** *Product of two MoGs is proportional to an MoG.*

*Proof.* Let $g_1(x) = \sum\limits_{i=1}^{K_1} \pi_{1,i} \mathcal{N}_x(\mu_{1,i}, \Sigma_{1,i})$ and $g_2(x) = \sum\limits_{j=1}^{K_2} \pi_{2,j} \mathcal{N}_x(\mu_{2,j}, \Sigma_{2,j})$ are two Mixtures of Gaussians. We have

$$g_1(x) \cdot g_2(x) = \sum_{i=1}^{K_1} \pi_{1,i} \mathcal{N}_x(\mu_{1,i}, \Sigma_{1,i}) \cdot \sum_{j=1}^{K_2} \pi_{2,j} \mathcal{N}_x(\mu_{2,j}, \Sigma_{2,j})$$
$$= \sum_{i=1,j=1}^{K_1, K_2} \pi_{1,i} \pi_{2,j} \mathcal{N}_x(\mu_{1,i}, \Sigma_{1,i}) \cdot \mathcal{N}_x(\mu_{2,j}, \Sigma_{2,j}) \tag{8.2}$$

By applying Lemma C.2 to Eq. (8.2), we have

$$g_1(x) \cdot g_2(x) = \sum_{i=1,j=1}^{K_1} \sum^{K_2} \pi_{1,i} \pi_{2,j} c_{ij} \mathcal{N}_x(\mu_{ij}, \Sigma_{ij})$$

$$= C \sum_{i=1,j=1}^{K_1} \sum^{K_2} \frac{\pi_{1,i} \pi_{2,j} c_{ij}}{C} \mathcal{N}_x(\mu_{ij}, \Sigma_{ij}) \qquad (8.3)$$

where $C = \sum_{i=1,j=1}^{K_1} \sum^{K_2} \pi_{1,i} \pi_{2,j} c_{ij}$. Clearly, Eq. (8.3) is proportional to an MoG with $K_1 \cdot K_2$ modes $\qquad \square$

**Theorem C.1.** $\prod_{t=1}^{T} g_t(x) = \prod_{t=1}^{T} \sum_{i=1}^{K} \pi_t^i g_t^i(x)$ *is a scaled MoG.*

*Proof.* By induction from Lemma C.3, we can easily show that product of $T$ MoGs is also proportional to an MoG. That means $\prod_{t=1}^{T} g_t(x)$ equals to a scaled MoG. $\qquad \square$

## C.4 Details of Data Descriptions and Model Implementations

Here we list all datasets used in our experiments:

- Open-domain datasets:

  - Cornell movie dialog: This corpus contains a large metadata-rich collection of fictional conversations extracted from 617 raw movies with 220,579 conversational exchanges between 10,292 pairs of movie characters. For each dialog, we preprocess the data by limiting the context length and the utterance output length to 20 and 10, respectively. The vocabulary is kept to top 20,000 frequently-used words in the dataset.

  - OpenSubtitles: This dataset consists of movie conversations in XML format. It also contains sentences uttered by characters in movies, yet it is much bigger and noisier than Cornell dataset. After preprocessing as above, there are more than 1.6 million pairs of contexts and utterance with chosen vocabulary of 40,000 words.

- Closed-domain datasets::

  - Live Journal (LJ) user question-answering dataset: question-answer dialog by LJ users who are members of anxiety, arthritis, asthma, autism,

depression, diabetes, and obesity LJ communities[1]. After preprocessing as above, we get a dataset of more than 112,000 conversations. We limit the vocabulary size to 20,000 most common words.

  – Reddit comments dataset: This dataset consists of posts and comments about movies in Reddit website[2]. A single post may have multiple comments constituting a multi-people dialog amongst the poster and commentors, which makes this dataset the most challenging one. We crawl over four millions posts from Reddit website and after preprocessing by retaining conversations whose utterance's length are less than 20, we have a dataset of nearly 200 thousand conversations with a vocabulary of more than 16 thousand words.

We trained with the following hyperparameters (according to the performance on the validate dataset): word embedding has size 96 and is shared across everywhere. We initialise the word embedding from Google's Word2Vec (Mikolov et al., 2013) pretrained word vectors. The hidden dimension of LSTM in all controllers is set to 768 for all datasets except the big OpenSubtitles whose LSTM dimension is 1024. The number of LSTM layers for every controller is set to 3. All the initial weights are sampled from a normal distribution with mean 0, standard deviation 0.1. The mini-batch size is chosen as 256. The models are trained end-to-end using the Adam optimiser (Kingma and Ba, 2014) with a learning rate of 0.001 and gradient clipping at 10. For models using memory, we set the number and the size of memory slots to 16 and 64, respectively. As indicated previously, it is not trivial to optimise VAE with RNN-like decoder due to the vanishing latent variable problem (Bowman et al., 2016). Hence, to make the variational models in our experiments converge we have to use the $KL$ annealing trick by adding to the $KL$ loss term an annealing coefficient $\alpha$ starts with a very small value and gradually increase up to 1.

---

[1]https://www.livejournal.com/
[2]https://www.reddit.com/r/movies/

## C.5   Full Reports on Model Performance

| Model | BLEU-1 | BLEU-2 | BLEU-3 | BLEU-4 | A-glove |
|---|---|---|---|---|---|
| Seq2Seq | 18.4 | 14.5 | 12.1 | 9.5 | 0.52 |
| Seq2Seq-att | 17.7 | 14.0 | 11.7 | 9.2 | 0.54 |
| DNC | 17.6 | 13.9 | 11.5 | 9.0 | 0.51 |
| CVAE | 16.5 | 13.0 | 10.9 | 8.5 | 0.56 |
| VLSTM | 18.6 | 14.8 | 12.4 | 9.7 | 0.59 |
| VMED (K=1) | 20.7 | 16.5 | 13.8 | 10.8 | 0.57 |
| VMED (K=2) | 22.3 | 18.0 | 15.2 | 11.9 | **0.64** |
| VMED (K=3) | 19.4 | 15.6 | 13.2 | 10.4 | 0.63 |
| VMED (K=4) | **23.1** | **18.5** | **15.5** | **12.3** | 0.61 |

Table C.1: Results on Cornell Movies

| Model | BLEU-1 | BLEU-2 | BLEU-3 | BLEU-4 | A-glove |
|---|---|---|---|---|---|
| Seq2Seq | 11.4 | 8.7 | 7.1 | 5.4 | 0.29 |
| Seq2Seq-att | 13.2 | 10.2 | 8.4 | 6.5 | 0.42 |
| DNC | 14.3 | 11.2 | 9.3 | 7.2 | 0.47 |
| CVAE | 13.5 | 10.2 | 8.4 | 6.6 | 0.45 |
| VLSTM | 16.4 | 12.7 | 10.4 | 8.1 | 0.43 |
| VMED (K=1) | 12.9 | 9.5 | 7.5 | 6.2 | 0.44 |
| VMED (K=2) | 15.3 | 13.8 | 10.4 | 8.8 | 0.49 |
| VMED (K=3) | **24.8** | **19.7** | **16.4** | **12.9** | **0.54** |
| VMED (K=4) | 17.9 | 14.2 | 11.8 | 9.3 | 0.52 |

Table C.2: Results on OpenSubtitles

| Model | BLEU-1 | BLEU-2 | BLEU-3 | BLEU-4 | A-glove |
|---|---|---|---|---|---|
| Seq2Seq | 13.1 | 10.1 | 8.3 | 6.4 | 0.45 |
| Seq2Seq-att | 11.4 | 8.7 | 7.1 | 5.6 | 0.49 |
| DNC | 12.4 | 9.6 | 7.8 | 6.1 | 0.47 |
| CVAE | 12.2 | 9.4 | 7.7 | 6.0 | 0.48 |
| VLSTM | 11.5 | 8.8 | 7.3 | 5.6 | 0.46 |
| VMED (K=1) | 13.7 | 10.7 | 8.9 | 6.9 | 0.47 |
| VMED (K=2) | 15.4 | 12.2 | 10.1 | 7.9 | **0.51** |
| VMED (K=3) | **18.1** | **14.8** | **12.4** | **9.8** | 0.49 |
| VMED (K=4) | 14.4 | 11.4 | 9.5 | 7.5 | 0.47 |

Table C.3: Results on LJ users question-answering

| Model | BLEU-1 | BLEU-2 | BLEU-3 | BLEU-4 | A-glove |
|---|---|---|---|---|---|
| Seq2Seq | 7.5 | 5.5 | 4.4 | 3.3 | 0.31 |
| Seq2Seq-att | 5.5 | 4.0 | 3.1 | 2.4 | 0.25 |
| DNC | 7.5 | 5.6 | 4.5 | 3.4 | 0.28 |
| CVAE | 5.3 | 4.3 | 3.6 | 2.8 | 0.39 |
| VLSTM | 6.9 | 5.1 | 4.1 | 3.1 | 0.27 |
| VMED (K=1) | 9.1 | 6.8 | 5.5 | 4.3 | 0.39 |
| VMED (K=2) | 9.2 | 7.0 | 5.7 | 4.4 | 0.38 |
| VMED (K=3) | **12.3** | **9.7** | **8.1** | **6.4** | **0.46** |
| VMED (K=4) | 8.6 | 6.9 | 5.9 | 4.6 | 0.41 |

Table C.4: Results on Reddit comments

# D   Supplementary for Chapter 6

## D.1   Derivation on the Bound Inequality in Linear Dynamic System

The linear dynamic system hidden state is described by the following recursive equation

$$h_t = Wx_t + Uh_{t-1} + b$$

By induction,

$$h_t = \sum_{i=1}^{t} U^{t-i}Wx_i + C$$

where $C$ is some constant with respect to $x_i$. In this case, $\frac{\partial h_t}{\partial x_i} = U^{t-i}W$. By applying norm sub-multiplicativity[3],

---

[3]If not explicitly stated otherwise, norm refers to any consistent matrix norm which satisfies sub-multiplicativity.

$$c_{i-1,t} = \left\| U^{t-i+1}W \right\|$$
$$\leq \|U\| \left\| U^{t-i}W \right\|$$
$$= \|U\| \, c_{i,t}$$

That is, $\lambda_c = \|U\|$.

## D.2   Derivation on the Bound Inequality in Standard RNN

The standard RNN hidden state is described by the following recursive equation

$$h_t = \tanh \left( W x_t + U h_{t-1} + b \right)$$

From $\frac{\partial h_t}{\partial x_i} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial x_i}$, by induction,

$$\frac{\partial h_t}{\partial x_i} = \left( \prod_{j=i+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_i}{\partial x_i} = \left( \prod_{j=i}^{t} diag \left( \tanh' \left( a_j \right) \right) \right) U^{t-i}W$$

where $a_j = W x_j + U h_{j-1} + b$ and $diag \left( \cdot \right)$ converts a vector into a diagonal matrix. As $0 \leq \tanh' \left( x \right) = 1 - \tanh \left( x \right)^2 \leq 1$, $\left\| diag \left( \tanh' \left( X \right) \right) \right\|$ is bounded by some value $B$. By applying norm sub-multiplicativity,

$$c_{i-1,t} = \left\| \left[ \prod_{j=i}^{t} diag \left( \tanh' \left( a_j \right) \right) \right] diag \left( \tanh' \left( a_{i-1} \right) \right) U^{t-i+1}W \right\|$$
$$\leq \|U\| \left\| \prod_{j=i}^{t} diag \left( \tanh' \left( a_j \right) \right) U^{t-i}W \right\| \left\| diag \left( \tanh' \left( a_{i-1} \right) \right) \right\|$$
$$= B \|U\| \, c_{i,t}$$

That is, $\lambda_c = B \|U\|$.

## D.3 Derivation on the Bound Inequality in LSTM

For the case of LSTM, the recursive equation reads

$$
\begin{aligned}
c_t &= \sigma \left( U_f x_t + W_f h_{t-1} + b_f \right) \odot c_{t-1} \\
&\quad + \sigma \left( U_i x_t + W_i h_{t-1} + b_i \right) \odot \tanh \left( U_z x_t + W_z h_{t-1} + b_z \right) \\
h_t &= \sigma \left( U_o x_t + W_o h_{t-1} + b_o \right) \odot \tanh \left( c_t \right)
\end{aligned}
$$

Taking derivatives,

$$
\begin{aligned}
\frac{\partial h_j}{\partial h_{j-1}} &= \sigma' \left( o_j \right) \tanh \left( c_j \right) W_o + \sigma \left( o_j \right) \tanh' \left( c_j \right) \sigma' \left( f_j \right) c_{j-1} W_f \\
&\quad + \sigma \left( o_j \right) \tanh' \left( c_j \right) \sigma' \left( i_j \right) \tanh \left( z_j \right) W_i + \sigma \left( o_j \right) \tanh' \left( c_j \right) \sigma \left( i_j \right) \tanh' \left( z_j \right) W_z \\
\frac{\partial h_{j-1}}{\partial x_{j-1}} &= \sigma' \left( o_{j-1} \right) \tanh \left( c_{j-1} \right) U_o + \sigma \left( o_{j-1} \right) \tanh' \left( c_{j-1} \right) \sigma' \left( f_{j-1} \right) c_{j-2} U_f \\
&\quad + \sigma \left( o_{j-1} \right) \tanh' \left( c_{j-1} \right) \sigma' \left( i_{j-1} \right) \tanh \left( z_{j-1} \right) U_i \\
&\quad + \sigma \left( o_{j-1} \right) \tanh' \left( c_{j-1} \right) \sigma \left( i_{j-1} \right) \tanh' \left( z_{j-1} \right) U_z \\
\frac{\partial h_j}{\partial x_j} &= \sigma' \left( o_j \right) \tanh \left( c_j \right) U_o + \sigma \left( o_j \right) \tanh' \left( c_j \right) \sigma' \left( f_j \right) c_{j-1} U_f \\
&\quad + \sigma \left( o_j \right) \tanh' \left( c_j \right) \sigma' \left( i_j \right) \tanh \left( z_j \right) U_i + \sigma \left( o_j \right) \tanh' \left( c_j \right) \sigma \left( i_j \right) \tanh' \left( z_j \right) U_z
\end{aligned}
$$

where $o_j$ denotes the value in the output gate at $j$-th timestep (similar notations are used for input gate $(i_j)$, forget gate $(f_j)$ and cell value $(z_j)$) and "non-matrix" terms actually represent diagonal matrices corresponding to these terms. Under the assumption that $h_0 = 0$, we then make use of the results stating that $\|c_t\|_\infty$ is bounded for all $t$ (Miller and Hardt, 2018) . By applying $l_\infty$-norm sub-multiplicativity and triangle inequality, we can show that

$$
\frac{\partial h_j}{\partial h_{j-1}} \frac{\partial h_{j-1}}{\partial x_{j-1}} = M \frac{\partial h_j}{\partial x_j} + N
$$

with

$$\|M\|_\infty \leq 1/4\,\|W_o\|_\infty + 1/4\,\|W_f\|_\infty\,\|c_j\|_\infty + 1/4\,\|W_i\|_\infty + \|W_z\|_\infty = B_m$$

$$\|N\|_\infty \leq 1/16\,\|W_oU_i\|_\infty + 1/16\,\|W_oU_f\|_\infty\left(\|c_j\|_\infty + \|c_{j-1}\|_\infty\right) + 1/4\,\|W_oU_z\|_\infty$$

$$+ 1/16\,\|W_iU_o\|_\infty + 1/16\,\|W_iU_f\|_\infty\left(\|c_j\|_\infty + \|c_{j-1}\|_\infty\right) + 1/4\,\|W_iU_z\|_\infty$$

$$+ 1/16\left(\|W_fU_o\|_\infty + 1/16\,\|W_fU_i\|_\infty + 1/4\,\|W_fU_z\|_\infty\right)\left(\|c_j\|_\infty + \|c_{j-1}\|_\infty\right)$$

$$+ 1/4\,\|W_zU_o\|_\infty + 1/4\,\|W_zU_f\|_\infty\left(\|c_j\|_\infty + \|c_{j-1}\|_\infty\right) + 1/4\,\|W_zU_i\|_\infty$$

$$= B_n$$

By applying $l_\infty$-norm sub-multiplicativity and triangle inequality,

$$
\begin{aligned}
c_{i-1,t} &= \left\|\prod_{j=i+1}^{t}\frac{\partial h_j}{\partial h_{j-1}}\frac{\partial h_i}{\partial h_{i-1}}\frac{\partial h_{i-1}}{\partial x_{i-1}}\right\|_\infty \\
&= \left\|\prod_{j=i+1}^{t}\frac{\partial h_j}{\partial h_{j-1}}\left(\frac{\partial h_i}{\partial x_i}m + n\right)\right\|_\infty \\
&\leq B_m c_{i,t} + B_n\prod_{j=i+1}^{t}\left\|\frac{\partial h_j}{\partial h_{j-1}}\right\|_\infty
\end{aligned}
$$

As LSTM is $\lambda$-contractive with $\lambda < 1$ in the $l_\infty$-norm (readers are recommended to refer to Miller and Hardt, (2018) for proof), which implies $\left\|\frac{\partial h_j}{\partial h_{j-1}}\right\|_\infty < 1$, $B_n\prod_{j=i+1}^{t}\left\|\frac{\partial h_j}{\partial h_{j-1}}\right\|_\infty \to 0$ as $t - i \to \infty$. For $t - i < \infty$, under the assumption that $\frac{\partial h_j}{\partial x_j} \neq 0$, we can always find some value $B < \infty$ such that $c_{i-1,t} \leq B c_{i,t}$. For $t - i \to \infty$, $\lambda_c \to B_m$. That is, $\lambda_c = \max(B_m, B)$.

## D.4   Proof of Theorem 6.1

*Proof.* Given that $\lambda_c c_{i,t} \geq c_{i-1,t}$ with some $\lambda_c \in \mathbb{R}^+$, we can use $c_{t,t}\lambda_c^{t-i}$ as the upper bound on $c_{i,t}$ with $i = \overline{1,t}$, respectively. Therefore,

$$f(0) \leq \sum_{t=1}^{T} c_{t,T} \leq c_{T,T}\sum_{t=1}^{T}\lambda_c^{T-t} = f(\lambda_c)$$

where $f(\lambda) = c_{T,T} \sum\limits_{t=1}^{T} \lambda^{T-t}$ is continuous on $\mathbb{R}^+$. According to intermediate value theorem, there exists $\lambda \in (0, \lambda_c]$ such that $c_{T,T} \sum\limits_{t=1}^{T} \lambda^{T-t} = \sum\limits_{t=1}^{T} c_{t,T}$. $\qquad\square$

## D.5   Proof of Theorem 6.2

*Proof.* According to Theorem 6.1, there exists some $\lambda_i \in \mathbb{R}^+$ such that the summation of contribution stored between $K_i$ and $K_{i+1}$ can be quantified as $c_{K_{i+1},K_{i+1}} \sum\limits_{t=K_i}^{K_{i+1}} \lambda_i^{K_{i+1}-t}$ (after ignoring contributions before $K_i$-th timestep for simplicity). Let denote $P(\lambda) = \sum\limits_{t=K_i}^{K_{i+1}} \lambda^{K_{i+1}-t}$, we have $P'(\lambda) > 0, \forall \lambda \in \mathbb{R}^+$. Therefore, $P(\lambda_i) \geq P\left(\min\limits_i (\lambda_i)\right)$. Let $C = \min\limits_i (c_{i,i})$ and $\lambda = \min\limits_i (\lambda_i)$, the average contribution stored in a MANN has a lower bound quantified as $I_\lambda$, where

$$I_\lambda = C \frac{\sum\limits_{t=1}^{K_1} \lambda^{K_1-t} + \sum\limits_{t=K_1+1}^{K_2} \lambda^{K_2-t} + ... + \sum\limits_{t=K_{D-1}+1}^{K_D} \lambda^{K_D-t} + \sum\limits_{t=K_D+1}^{T} \lambda^{T-t}}{T} \tag{8.4}$$

. $\qquad\square$

## D.6   Proof of Theorem 6.3

*Proof.* The second-order derivative of $f_\lambda(x)$ reads

$$f_\lambda''(x) = -\frac{(\ln \lambda)^2}{1 - \lambda} \lambda^x \tag{8.5}$$

We have $f_{\lambda,}''(x) \leq 0$ with $\forall x \in \mathbb{R}^+$ and $1 > \lambda > 0$, so $f_\lambda(x)$ is a concave function. Thus, we can apply Jensen inequality as follows:

$$\frac{1}{D+1} \sum_{i=1}^{D+1} f_\lambda(l_i) \leq f_\lambda \left( \sum_{i=1}^{D+1} \frac{1}{D+1} l_i \right) = f_\lambda \left( \frac{T}{D+1} \right) \tag{8.6}$$

Equality holds if and only if $l_1 = l_2 = ... = l_{D+1} = \frac{T}{D+1}$. We refer to this as *Uniform Writing* strategy. By plugging the optimal values of $l_i$, we can derive the maximised average contribution as follows,

$$I_\lambda max \equiv g_\lambda\left(T, D\right) = \frac{C\left(D + 1\right)}{T}\left(\frac{1 - \lambda^{\frac{T}{D+1}}}{1 - \lambda}\right) \tag{8.7}$$

When $\lambda = 1$, $I_\lambda = \frac{C}{T}\sum_{i=1}^{D+1} l_i = C$. This is true for all writing strategies. Thus, Uniform Writing is optimal for $0 < \lambda \leq 1$. $\qquad\square$

We can show that this solution is also optimal for the case $\lambda > 1$. As $f_\lambda''\left(x\right) > 0$ with $\forall x \in \mathbb{R}^+; \lambda > 1$, $f_\lambda\left(x\right)$ is a convex function and Eq. (8.6) flips the inequality sign. Thus, $I_\lambda$ reaches its minimum with Uniform Writing. For $\lambda > 1$, minimising $I_\lambda$ is desirable to prevent the system from diverging.

We can derive some properties of function $g$. Let $x = \frac{D+1}{L}, g_\lambda\left(L, D\right) = g_\lambda\left(x\right) = Cx\left(\frac{\lambda^{\frac{1}{x}} - 1}{\lambda - 1}\right)$. We have $g_\lambda'\left(x\right) = C\lambda\left(1 - \lambda^{\frac{1}{x}}\right)\left(x - \ln\lambda\right) > 0$ with $0 < \lambda \leq 1, \forall x \geq 0$, so $g_\lambda\left(T, D\right)$ is an increasing function if we fix $T$ and let $D$ vary. That explains why having more memory slots helps improve memorisation capacity. If $D = 0$, $g_\lambda\left(T, 0\right)$ becomes E.q (6.5). In this case, MANNs memorisation capacity converges to that of recurrent networks.

## D.7   Summary of Synthetic Discrete Task Format

| Task | Input | Output |
|---|---|---|
| Double | $x_1 x_2...x_T$ | $x_1 x_2...x_T x_1 x_2...x_T$ |
| Copy | $x_1 x_2...x_T$ | $x_1 x_2...x_T$ |
| Reverse | $x_1 x_2...x_T$ | $x_T x_{T-1}...x_1$ |
| Add | $x_1 x_2...x_T$ | $\frac{x_1 + x_{T-1}}{2} \frac{x_2 + x_{T-2}}{2}...\frac{x_{\lfloor T/2 \rfloor} + x_{\lceil T/2 \rceil}}{2}$ |
| Max | $x_1 x_2...x_T$ | $\max\left(x_1, x_2\right)\max\left(x_3, x_4\right)...\max\left(x_{T-1}, x_T\right)$ |

Table D.1: Synthetic discrete task's input-output formats. $T$ is the sequence length.

## D.8   UW Performance on Bigger Memory

| Model | $N_h$ | Copy (L=500) |
|---|---|---|
| DNC | 128 | 24.19% |
| DNC+UW | 128 | 81.45% |

Table D.2: Test accuracy (%) on synthetic copy task. MANNs have 50 memory slots. Both models are trained with 100,000 mini-batches of size 32.

## D.9  Memory Operating Behaviors on Synthetic Tasks

In this section, we pick three models (DNC, DNC+UW and DNC+CUW) to anal-
yse their memory operating behaviors. Fig. D.1 visualises the values of the write
weights and read weights for the copy task during encoding input and decoding
output sequence, respectively. In the copy task, as the sequence length is 50 while
the memory size is 4, one memory slot should contain the accumulation of multiple
timesteps. This principle is reflected in the decoding process in three models, in
which one memory slot is read repeatedly across several timesteps. Notably, the
number of timesteps consecutively spent for one slot is close to 10-the optimal in-
terval, even for DNC ( Fig. D.1(a)), which implies that the ultimate rule would
be the uniform rule. As UW and CUW are equipped with uniform writing, their
writing patterns follow the rule perfectly. Interestingly, UW chooses the first writ-
ten location for the final write (corresponding to the <eos> token) while CUW
picks the last written location. As indicated in Figs. D.1(b) and (c), both of them
can learn the corresponding reading pattern for decoding process, which leads to
good performances. On the other hand, regular DNC fails to learn a perfect writing
strategy. Except for the timesteps at the end of the sequence, the timesteps are
distributed to several memory slots while the reading phase attends to one memory
slot repeatedly. This explains why regular DNC cannot compete with the other two
proposed methods in this task.

For the max task, Fig. D.2 displays similar visualisation with an addition of write
gate during encoding phase. The write gate indicates how much the model should
write the input at some timestep to the memory. A zero write gate means there
is no writing. For this task, a good model should discriminate between timesteps
and prefer writing the greater ones. As clearly seen in Fig. D.2(a), DNC suffers the
same problem as in copy task, unable to synchronise encoding writing with decoding
reading. Also, DNC's write gate pattern does not show reasonable discrimination.
For UW (Fig. D.2(b)), it tends to write every timestep and relies on uniform
writing principle to achieve write/read accordance and thus better results than DNC.
Amongst all, CUW is able to ignore irrelevant timesteps and follows uniform writing
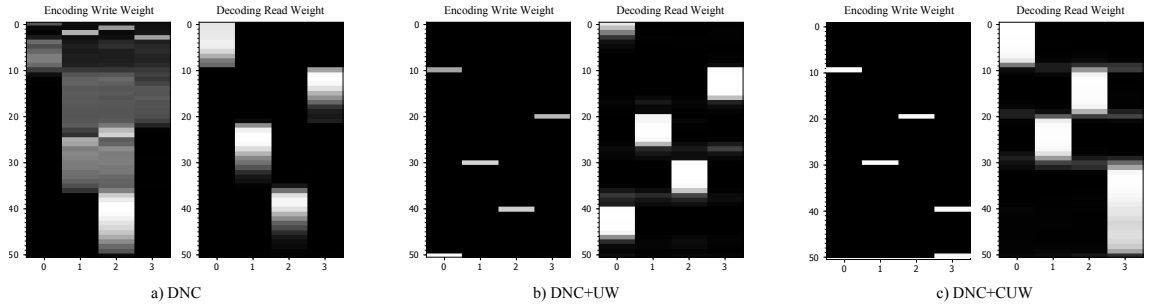at the same time (see Fig. D.2(c)).

Figure D.1: Memory operations on copy task in DNC (a), DNC+UW (b) and DNC+CUW(c). Each row is a timestep and each column is a memory slot.
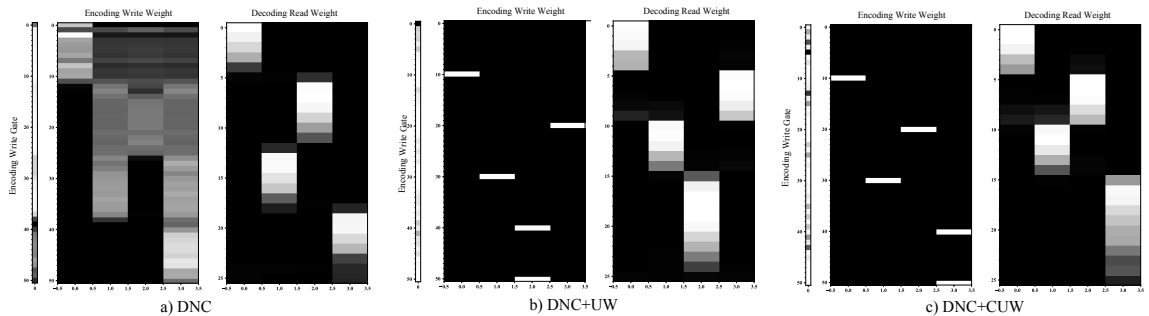


Figure D.2: Memory operations on max task in DNC (a), DNC+UW (b) and DNC+CUW(c). Each row is a timestep and each column is a memory slot.

## D.10   Visualisations of Model Performance on Sinusoidal Regression Tasks

We pick randomly 3 input sequences and plot the output sequences produced by DNC, UW and CUW in Figs. D.3 (clean) and D.4 (noisy). In each plot, the first and last 100 timesteps correspond to the given input and generated output, respectively. The ground truth sequence is plotted in red while the predicted in blue. We also visualise the values of MANN write gates through time in the bottom of each plots. In irregular writing encoding phase, the write gate is computed even when there is no write as it reflects how much weight the controller puts on the timesteps. In decoding, we let MANNs write to memory at every timestep to allow instant update of memory during inference.

Under clean condition, all models seem to attend more to late timesteps during encoding, which makes sense as focusing on late periods of sine wave is enough for later reconstruction. However, this pattern is not clear in DNC and UW as in CUW. During decoding, the write gates tend to oscillate in the shape of sine wave, which is also a good strategy as this directly reflects the amplitude of generation target. In this case, both UW and CUW demonstrate this behavior clearer than DNC.
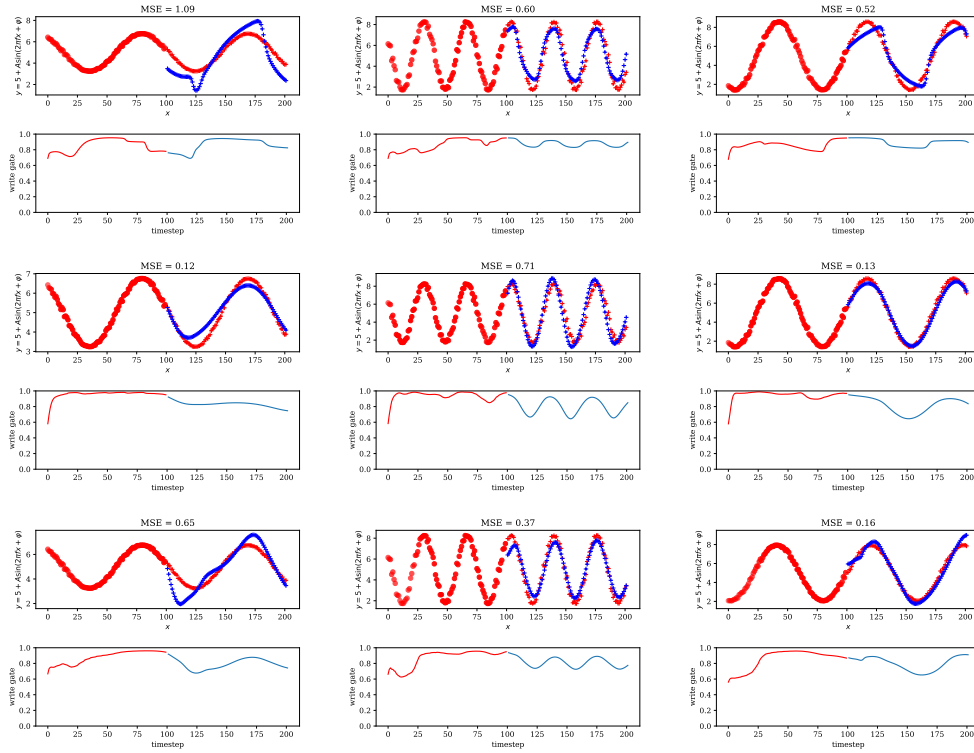
Figure D.3: Sinusoidal generation with clean input sequence for DNC, UW and CUW in top-down order.

Under noisy condition, DNC and CUW try to follow sine-shape writing strategy. However, only CUW can learn the pattern and assign write values in accordance with the signal period, which helps CUW decoding achieve highest accuracy. On the other hand, UW choose to assign write value equally and relies only on its maximisation of timestep contribution. Although it achieves better results than DNC, it underperforms CUW.

## D.11    Comparison with Non-Recurrent Methods in Flatten Image Classification Task

| Model | MNIST | pMNIST |
|---|---|---|
| The Transformer[⋆] | 98.9 | 97.9 |
| Dilated CNN[♦] | 98.3 | 96.7 |
| DNC+CUW | 99.1 | 96.3 |

Table D.3: Test accuracy (%) on MNIST, pMNIST. Previously reported results are from Vaswani et al., (2017)[⋆] and Chang et al., (2017)[♦].
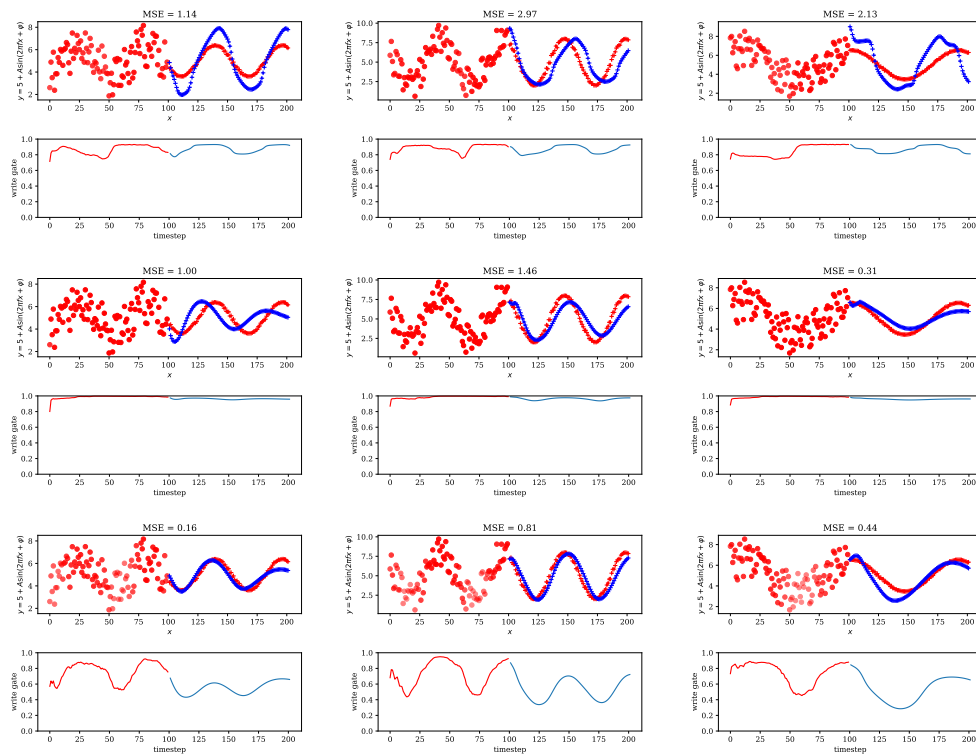
Figure D.4: Sinusoidal generation with noisy input sequence for DNC, UW and CUW in top-down order.

## D.12    Details on Document Classification Datasets

| Dataset | Classes | Average lengths | Max lengths | Train samples | Test samples |
|---|---|---|---|---|---|
| IMDb | 2 | 282 | 2,783 | 25,000 | 25,000 |
| Yelp Review Polarity (Yelp P.) | 2 | 156 | 1,381 | 560,000 | 38,000 |
| Yelp Review Full (Yelp F.) | 5 | 158 | 1,381 | 650,000 | 50,000 |
| AG's News (AG) | 4 | 44 | 221 | 120,000 | 7,600 |
| DBPedia (DBP) | 14 | 55 | 1,602 | 560,000 | 70,000 |
| Yahoo! Answers (Yah. A.) | 10 | 112 | 4,392 | 1,400,000 | 60,000 |

Table D.4: Statistics on several big document classification datasets

## D.13 Document Classification Detailed Records

| | Model | AG | IMDb | Yelp P. | Yelp F. |
|---|---|---|---|---|---|
| | 1 | 93.42 | **91.39** | **96.39** | 64.89 |
| UW | 2 | 93.52 | 91.30 | 96.31 | 64.97 |
| | 3 | **93.69** | 91.25 | 96.39 | **65.26** |
| | Mean/Std | 93.54±0.08 | 91.32±0.04 | 96.36±0.03 | 65.04±0.11 |
| | 1 | 93.61 | 91.26 | **96.42** | **65.63** |
| CUW | 2 | **93.87** | 91.18 | 96.29 | 65.05 |
| | 3 | 93.70 | **91.32** | 96.36 | 64.80 |
| | Mean/Std | 93.73±0.08 | 91.25±0.04 | 96.36±0.04 | 65.16±0.24 |

Table D.5: Document classification accuracy (%) on several datasets reported for 3 different runs. Bold denotes the best records.

# E Supplementary for Chapter 7

## E.1 Full Learning Curves on Single NTM Tasks
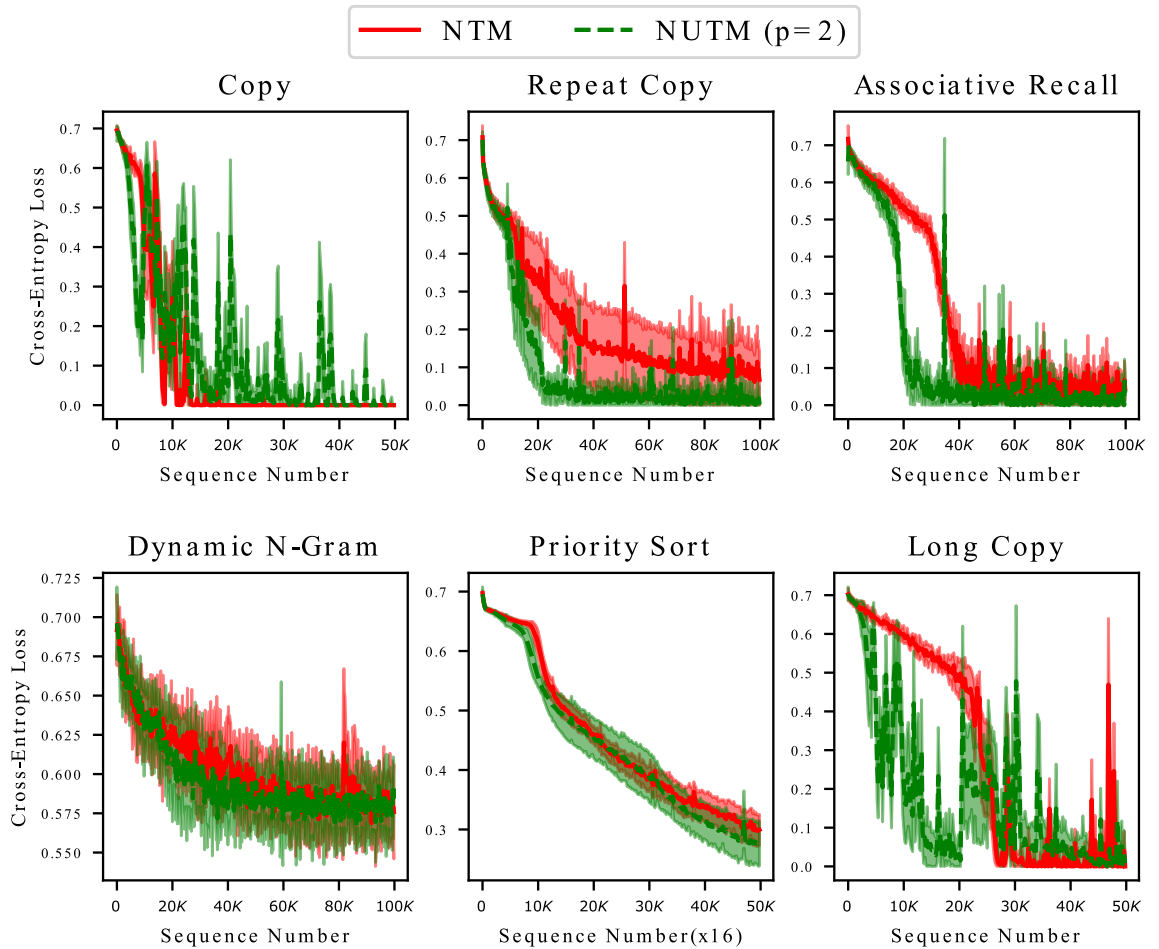


Figure E.1: Learning curves on NTM tasks.

## E.2 Clustering on The Latent Space

As previously mentioned in Sec. 3.3, MANN should let its states form clusters to well-simulate Turing Machine. Fig. E.2 (a) and (c) show NTM actually organises its $c_t$ space into clusters corresponding to processing states (e.g, encoding and decoding). NUTM, which explicitly partitions this space, clearly learn better clusters of $c_t$ (see Fig. E.2 (b) and (d)). This contributes to NUTM's outperformance over NTM.
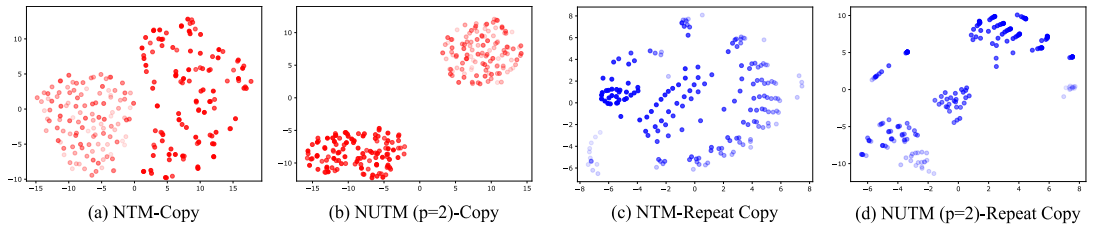
Figure E.2: Visualisation of the first two principal components of $c_t$ space in NTM (a,c) and NUTM (b,d) for Copy (red) and Repeat Copy (blue). Fader color denotes lower timestep in a sequence. Both can learn clusters of hidden states yet NUTM exhibits clearer partition.

## E.3 Program Usage Visualisations

E.3.1 and E.3.2 visualise the best inferences of NUTM on test data from single and sequencing tasks. Each plot starts with the input sequence and the predicted output sequence with error bits in the first row. The second and fourth rows depict the read and write locations on data memory, respectively. The third and fifth rows depict the program distribution of the read head and write head, respectively. E.3.3 visualises random failed predictions of NTM on sequencing tasks. The plots follow previous pattern except for the program distribution rows.

### E.3.1 Visualisation on Program Distribution across Timesteps (Single Tasks)
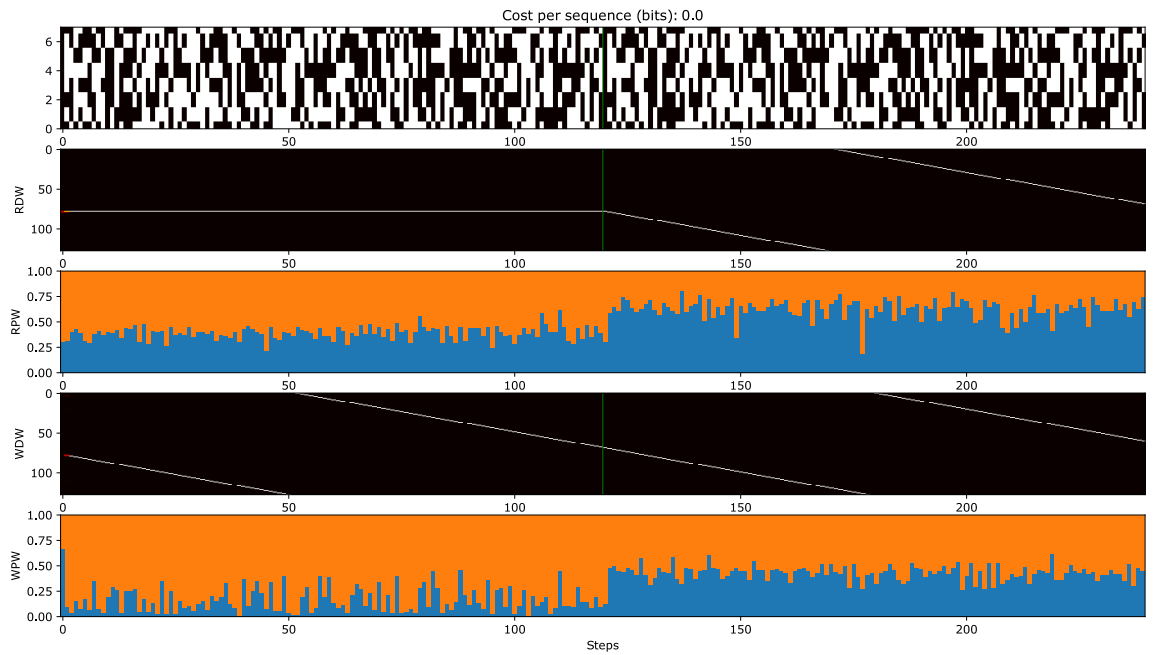


Figure E.3: Copy (p=2).
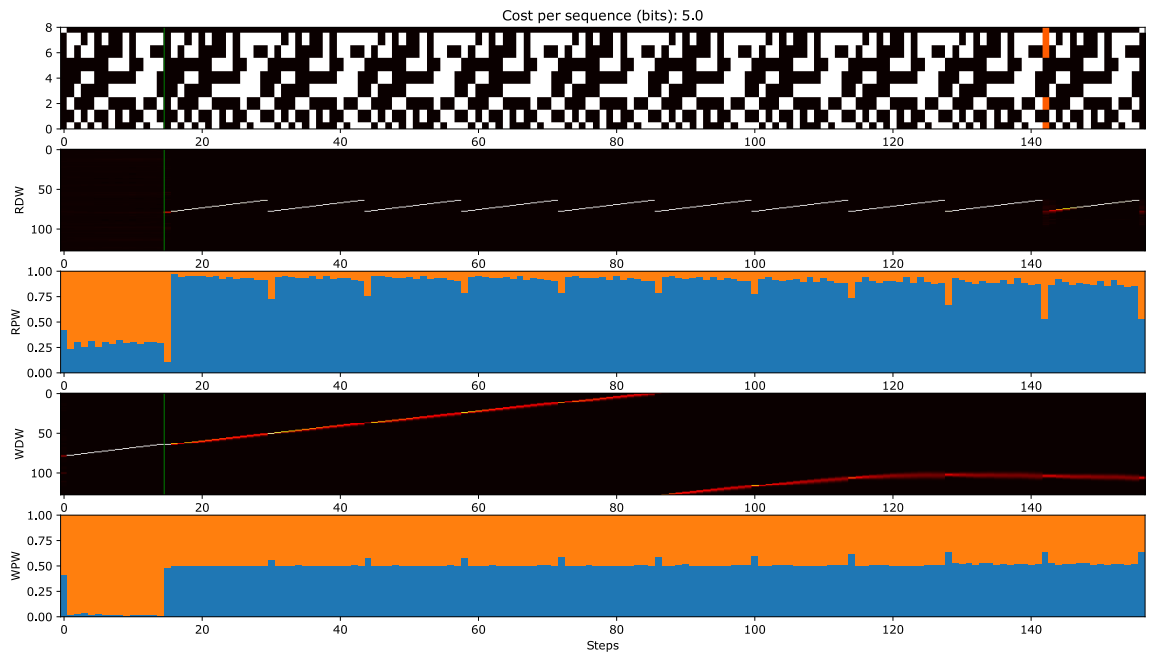


Figure E.4: Repeat Copy (p=2).
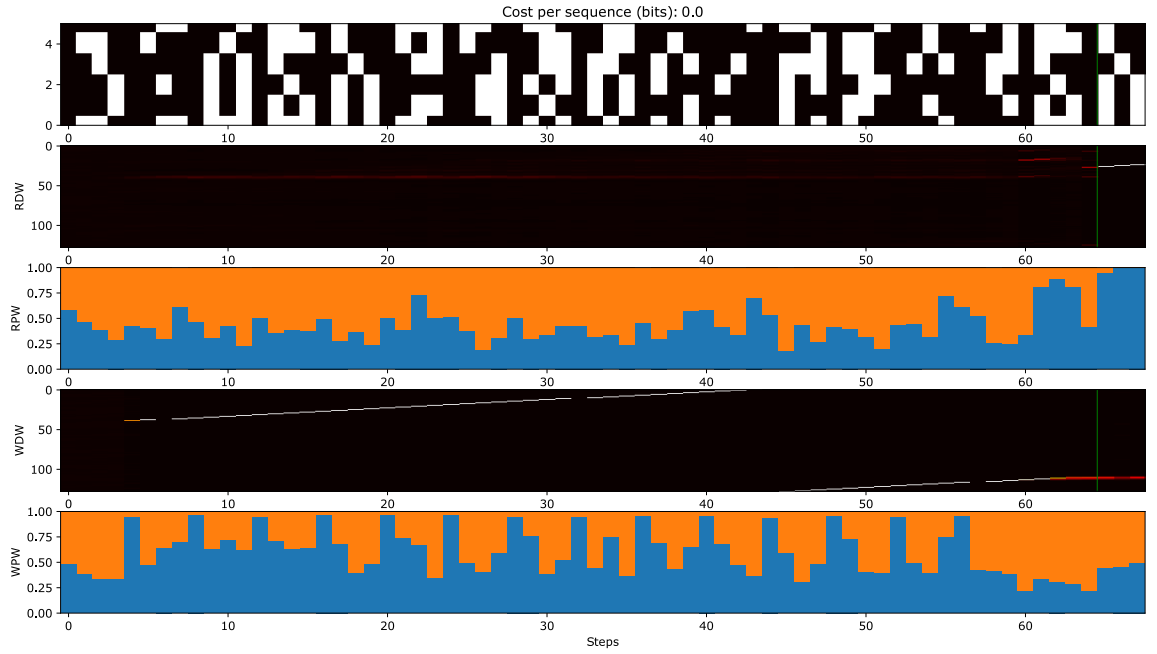
Figure E.5: Associative Recall (p=2).
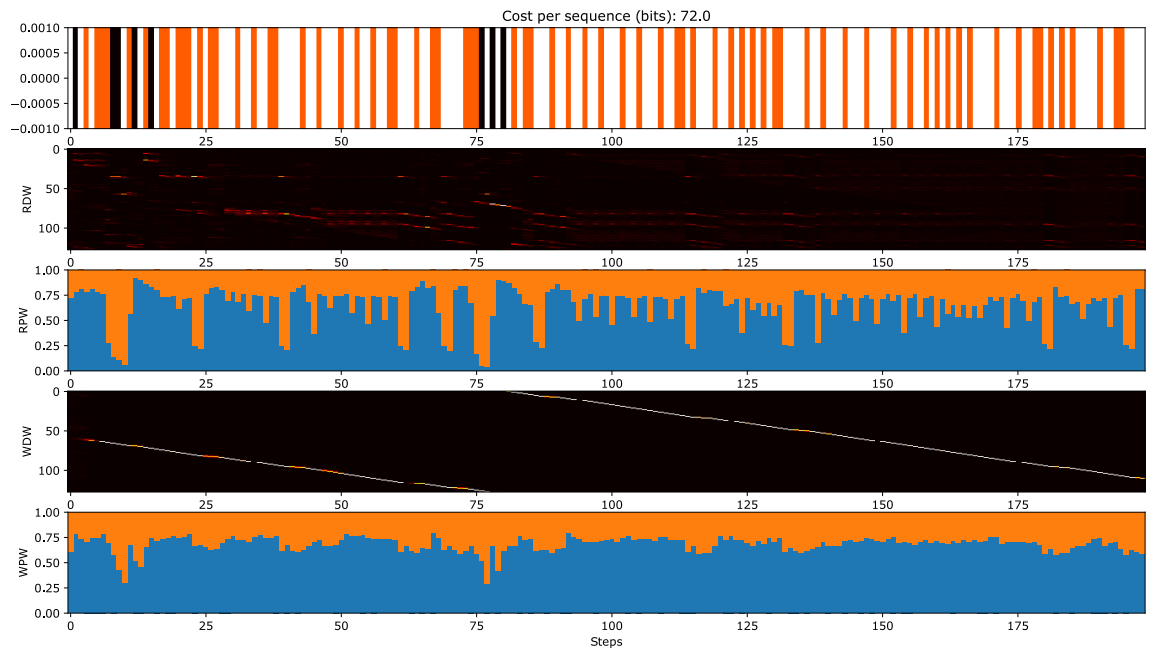


Figure E.6: Dynamic N-grams (p=2).
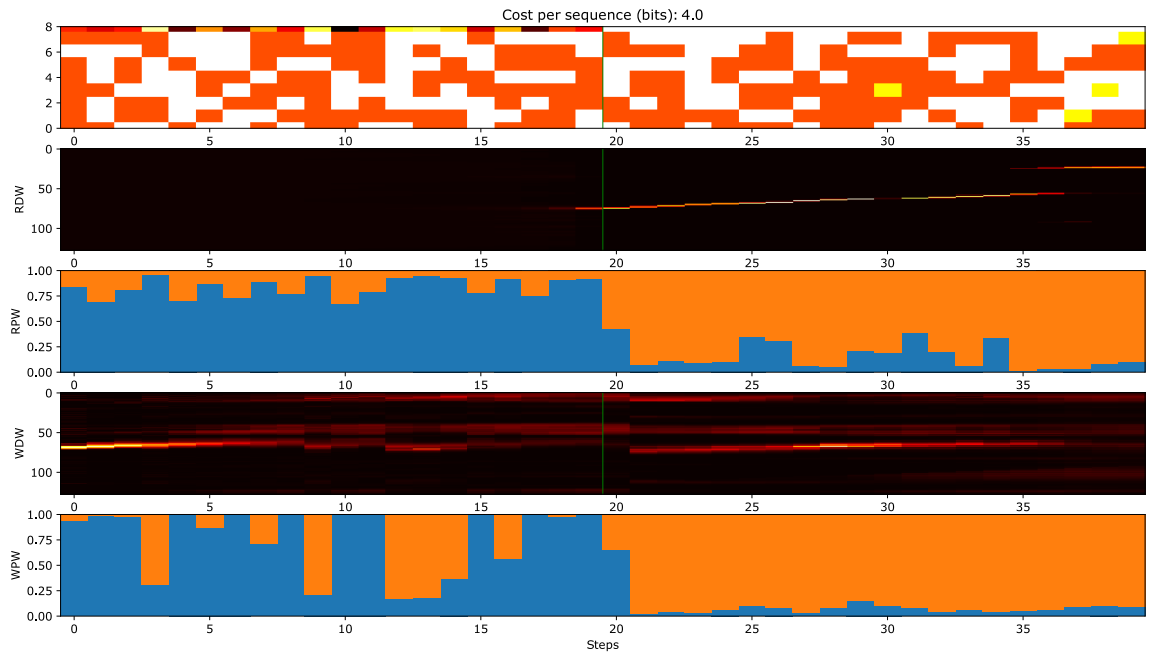
Figure E.7: Priority Sort (p=2).
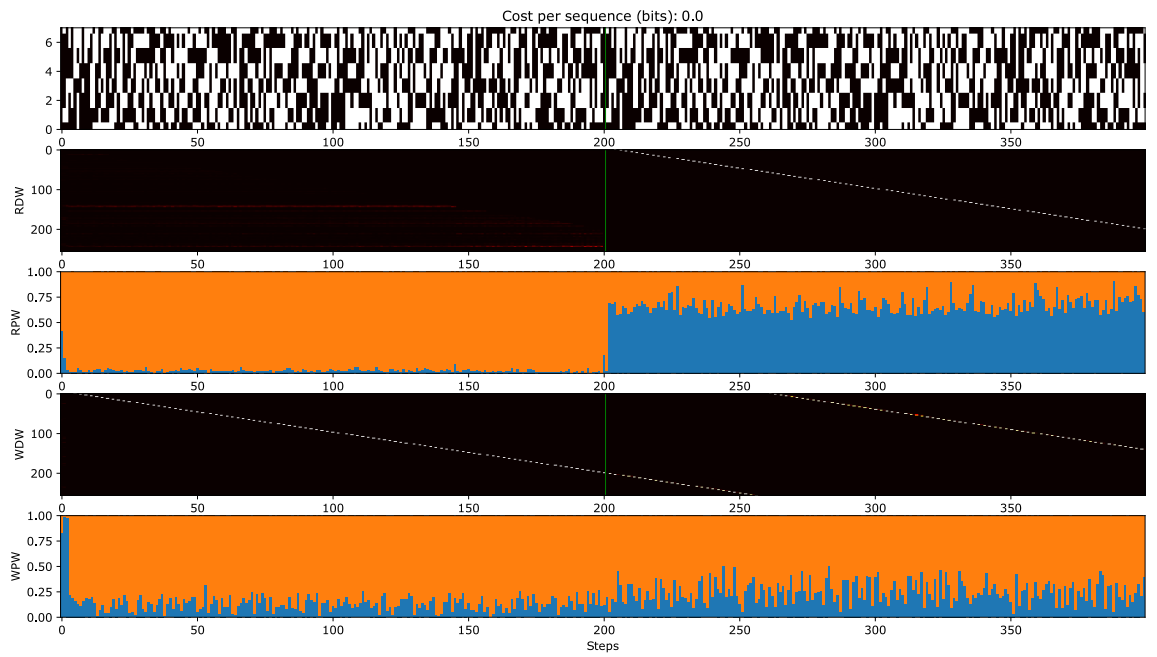


Figure E.8: Long Copy (p=2).

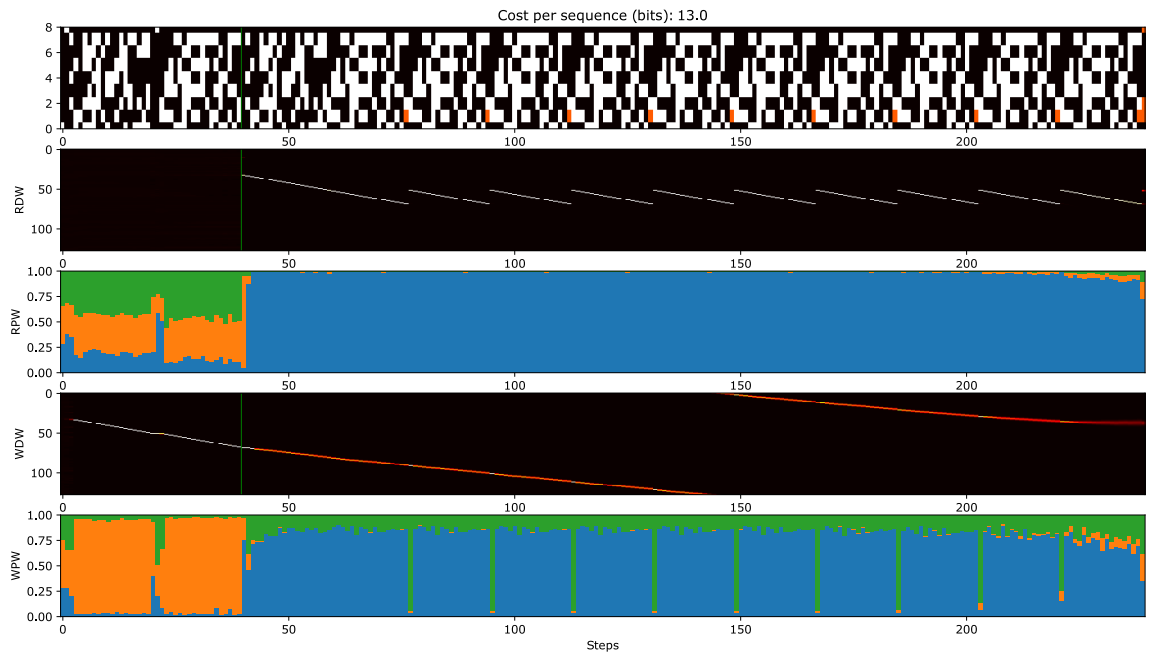### E.3.2 Visualisation on Program Distribution across Timesteps (Sequencing Tasks)
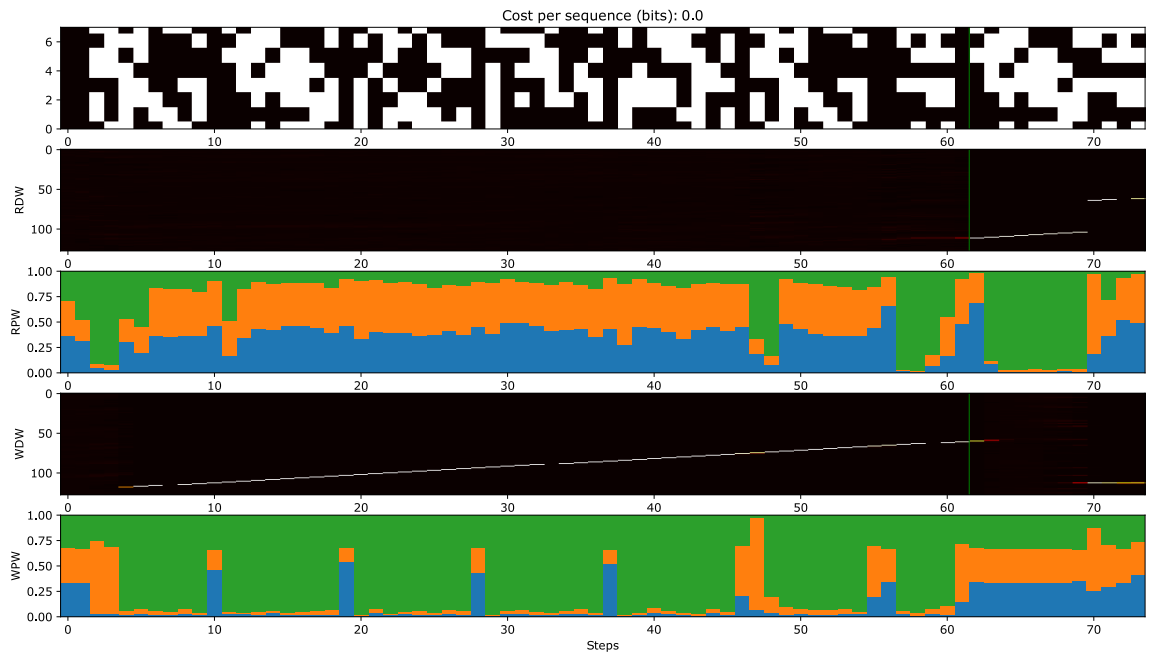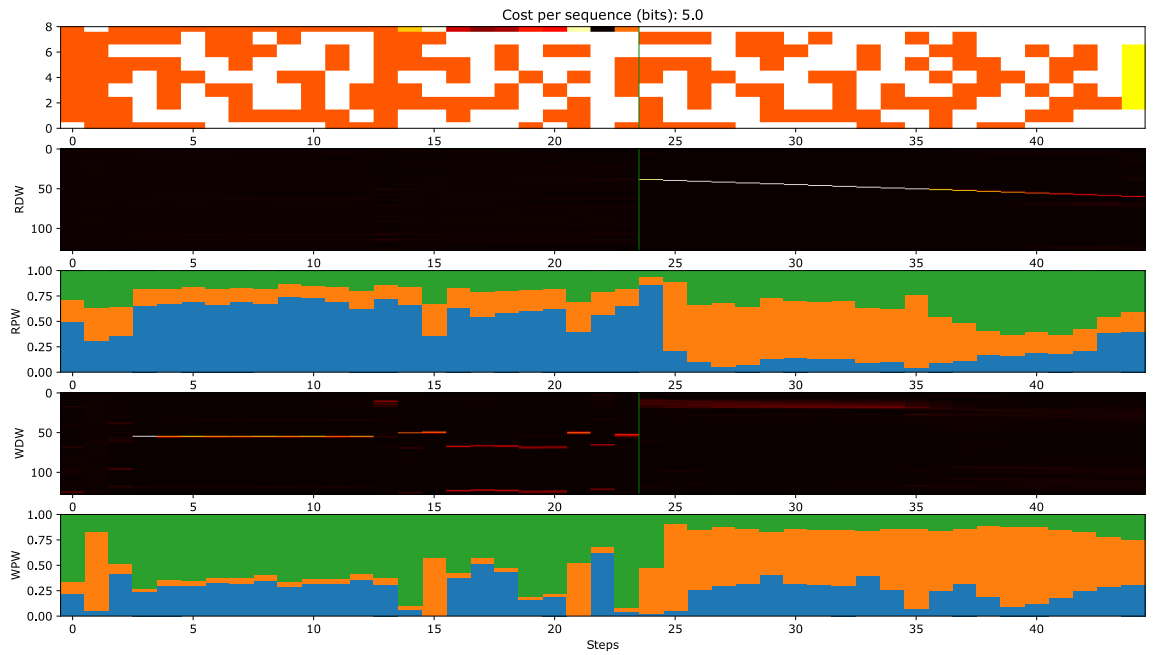


Figure E.9: Copy+Repeat Copy (p=3).
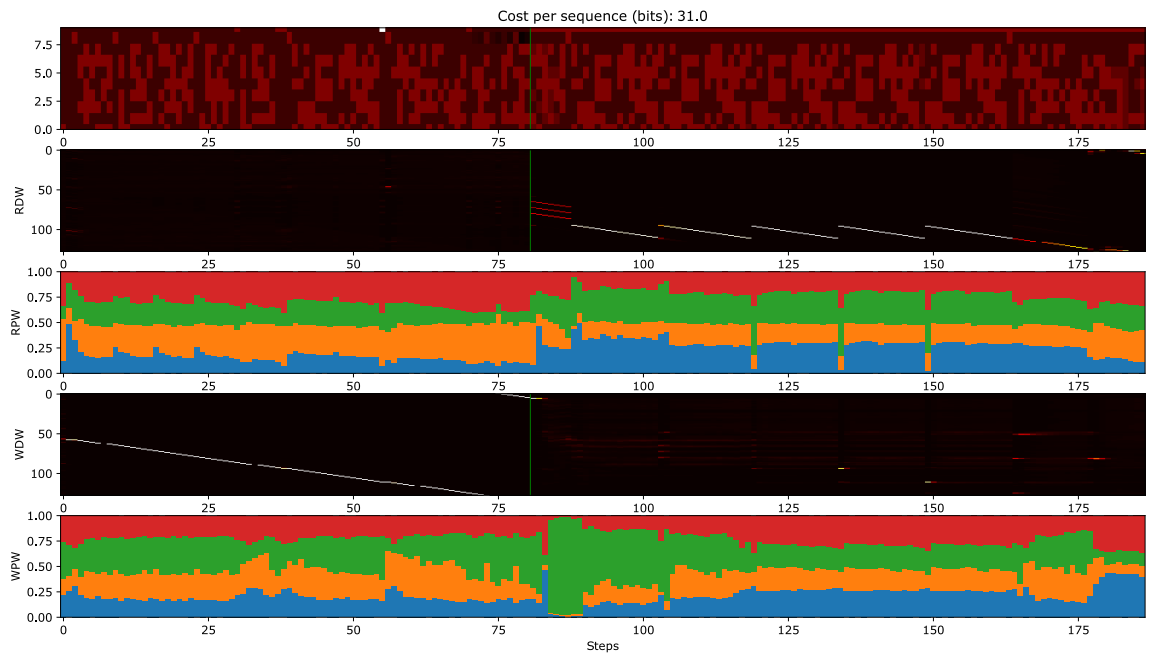


Figure E.10: Copy+Associative Recall (p=3).

Figure E.11: Copy+Priority Sort (p=3).



Figure E.12: Copy+Repeat Copy+Associative Recall+Priority Sort (p=4).

### E.3.3 Perseveration Phenomenon in NTM (Sequencing Tasks)
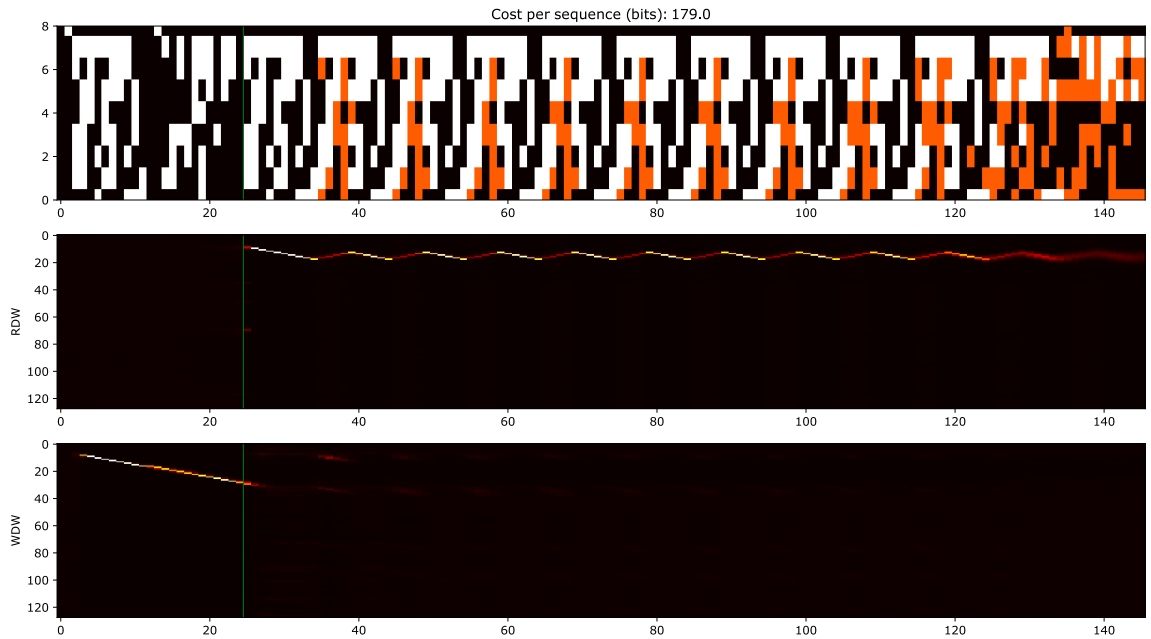


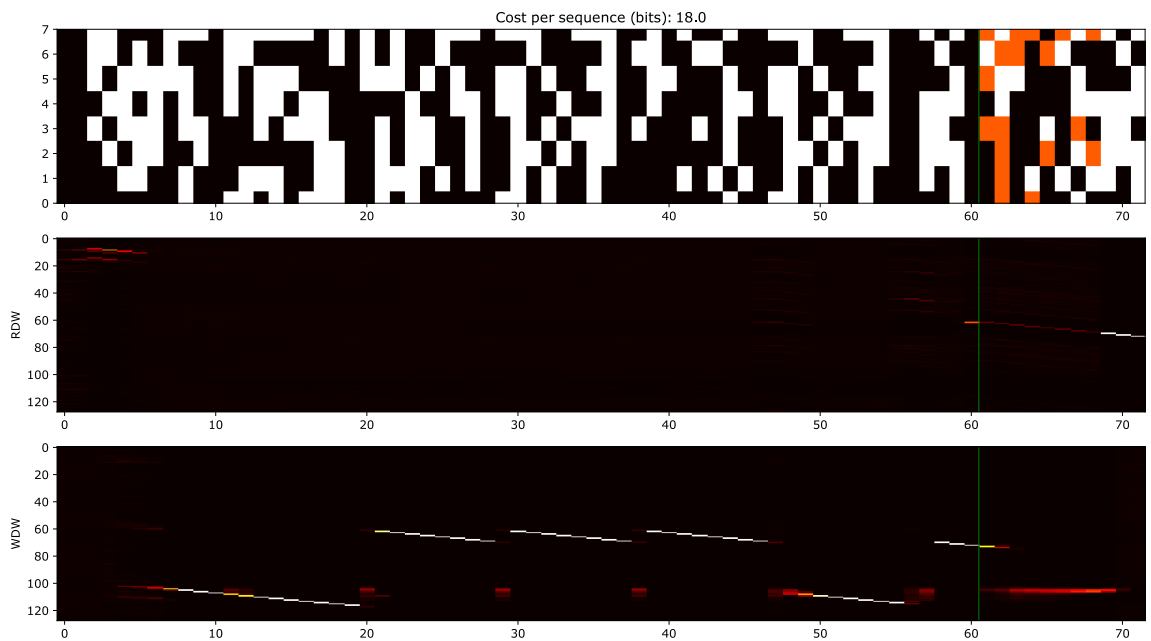Figure E.13: Copy+Repeat Copy perseveration (only Repeat Copy).



Figure E.14: Copy+Associative Recall perseveration (only Copy).
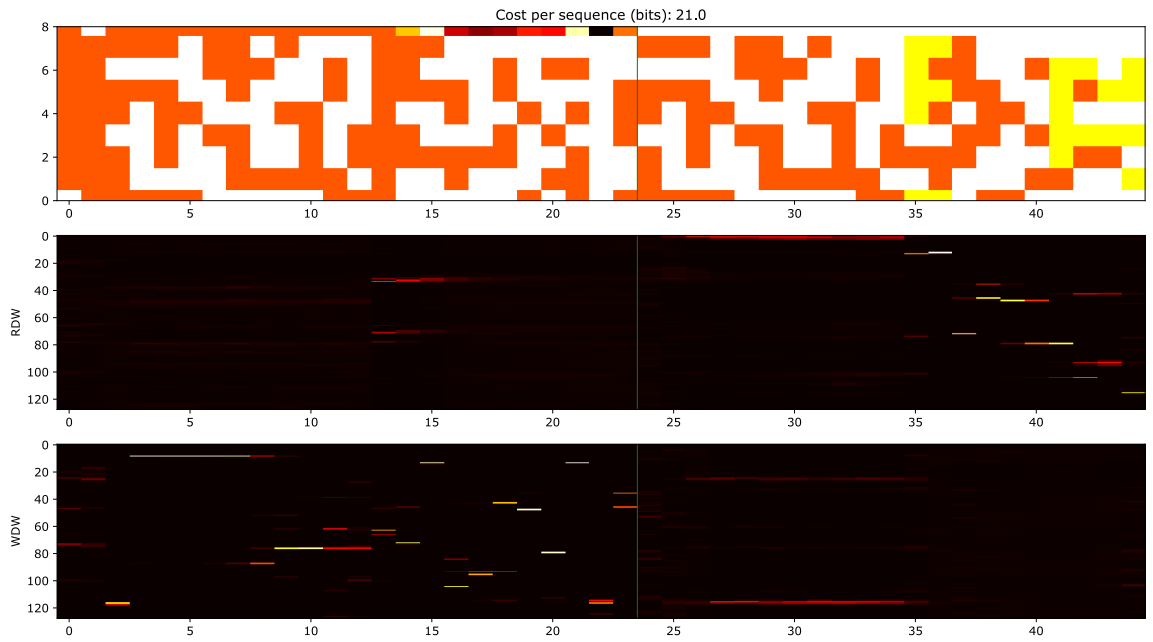
Figure E.15: Copy+Priority Sort perseveration (only Copy).
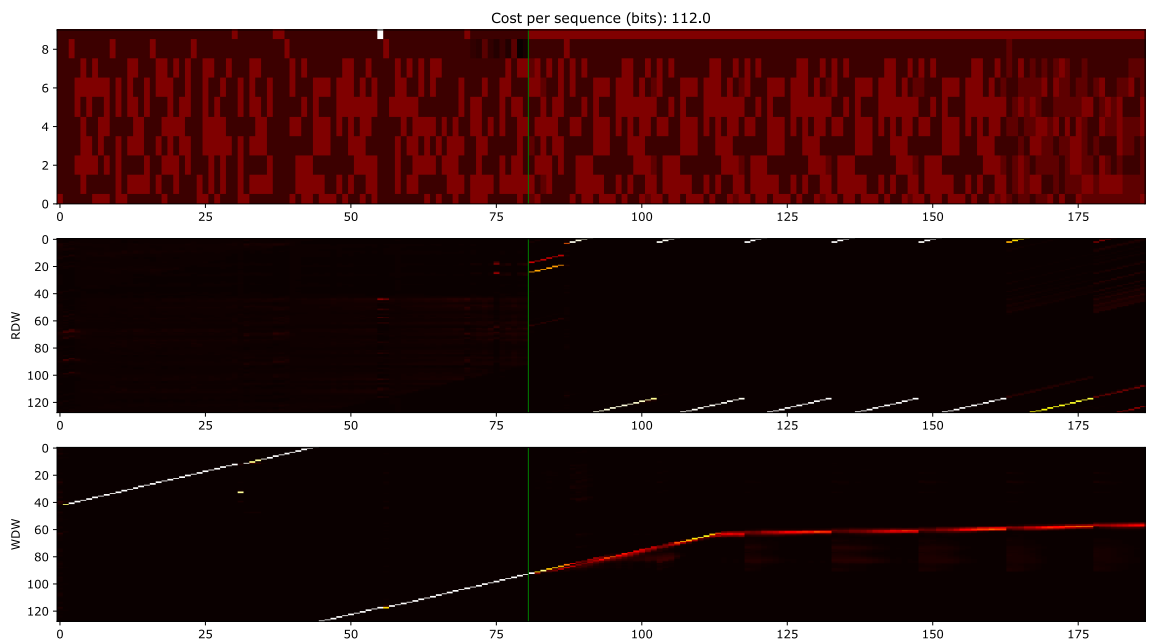


Figure E.16: Copy+Repeat Copy+Associative Recall+Priority Sort perseveration (only Repeat Copy).

## E.4 Details on Synthetic Tasks

### E.4.1 NTM Single Tasks

| Tasks | #Head | | Controller Size | | Memory Size | | #Parameters | |
|---|---|---|---|---|---|---|---|---|
| | NTM | NUTM | NTM | NUTM | NTM | NUTM | NTM | NUTM |
| Copy | 1 | 1 | 100 | 80 | 128 | 128 | 63,260 | 52,206 |
| Repeat Copy | 1 | 1 | 100 | 80 | 128 | 128 | 63,381 | 52,307 |
| Associative Recall | 1 | 1 | 100 | 80 | 128 | 128 | 62,218 | 51,364 |
| Dynamic N-grams | 1 | 1 | 100 | 80 | 128 | 128 | 58,813 | 48,619 |
| Priority Sort | 5 | 5 | 200 | 150 | 128 | 128 | 344,068 | 302,398 |
| Long Copy | 1 | 1 | 100 | 80 | 256 | 256 | 63,260 | 52,206 |

Table E.1: Model hyper-parameters (single tasks).

| Tasks | Training | Testing |
|---|---|---|
| Copy | Sequence length range: [1, 20] | Sequence length: 120 |
| Repeat Copy | Sequence length range: [1, 10] #Repeat range: [1, 10] | Sequence length range: [10, 20] #Repeat range: [10, 20] |
| Associative Recall | Sequence length: 3 #Item range: [2, 6] Item length: 3 | Sequence length: 3 #Item range: [6, 20] Item length: 3 |
| Dynamic N-grams | Sequence length: 50 | Sequence length: 200 |
| Priority Sort | #Item: 20 #Sorted Item: 16 | #Item: 20 #Sorted Item: 20 |
| Long Copy | Sequence length range: [1, 40] | Sequence length: 200 |

Table E.2: Task settings (single tasks).

### E.4.2 NTM Sequencing Tasks

| Tasks | #Head | | Controller Size | | Memory Size | | #Parameters | |
|---|---|---|---|---|---|---|---|---|
| | NTM | NUTM | NTM | NUTM | NTM | NUTM | NTM | NUTM |
| C+RC | 1 | 1 | 200 | 150 | 128 | 128 | 206,481 | 153,941 |
| C+AR | 1 | 1 | 200 | 150 | 128 | 128 | 206,260 | 153,770 |
| C+PS | 3 | 3 | 200 | 150 | 128 | 128 | 275,564 | 263,894 |
| C+RC+AR+PS | 3 | 3 | 250 | 200 | 128 | 128 | 394,575 | 448,379 |

Table E.3: Model hyper-parameters (sequencing tasks).

| Tasks | Training | Testing |
|---|---|---|
| C+RC | Sequence length range: [1, 10] | Sequence length range: [10, 20] |
| | #Repeat range: [1, 10] | #Repeat range: [10, 15] |
| C+AR | Sequence length range: [1, 10] | Sequence length range: [10, 20] |
| | #Item range: [2, 4] | #Item range: [4, 6] |
| | Item length: 8 | Item length: 8 |
| C+PS | Sequence length range: [1, 10] | Sequence length range: [10, 20] |
| | #Item: 10 | #Item: 10 |
| | #Sorted Item: 8 | #Sorted Item: 10 |
| C+RC+AR+PS | Sequence length range: [1, 10] | Sequence length range: [10, 20] |
| | #Repeat range: [1, 5] | #Repeat: 6 |
| | #Item range: [2, 4] | #Item: 5 |
| | Item length: 6 | Item length: 6 |
| | #Item: 10 | #Item: 10 |
| | #Sorted Item: 8 | #Sorted Item: 10 |

Table E.4: Task settings (sequencing tasks).

### E.4.3 Continual Procedure Learning Tasks

| Tasks | Training | Testing |
|---|---|---|
| Copy | Sequence length range: [1, 10] | Sequence length range: [1, 10] |
| Repeat Copy | Sequence length range: [1, 5] | Sequence length range: [1, 5] |
| | #Repeat range: [1, 5] | #Repeat range: [1, 5] |
| Associative Recall | Sequence length: 3 | Sequence length: 3 |
| | #Item range: [2, 3] | #Item range: [2, 3] |
| | Item length: 3 | Item length: 3 |
| Priority Sort | #Item: 10 | #Item: 10 |
| | #Sorted Item: 8 | #Sorted Item: 8 |

Table E.5: Task settings (continual procedure learning tasks).

## E.5 Details on Few-Shot Learning Task

We use similar hyper-parameters as in (Santoro et al., 2016), which are reported in Tab. E.6.

| Model | $p$ | #Head | Controller Size | $N$ | $M$ | $\mathbf{M}_p.K$ Size | Optimiser | Learning Rate |
|---|---|---|---|---|---|---|---|---|
| MANN (LRUA) | 1 | 4 | 200 | 128 | 40 | 0 | RMSprop | $10^{-4}$ |
| NUTM (LRUA) | 2 | 4 | 180 | 128 | 40 | 2 | RMSprop | $10^{-4}$ |
| NUTM (LRUA) | 3 | 4 | 150 | 128 | 40 | 3 | RMSprop | $10^{-4}$ |

Table E.6: Hyper-parameters for few-shot learning.

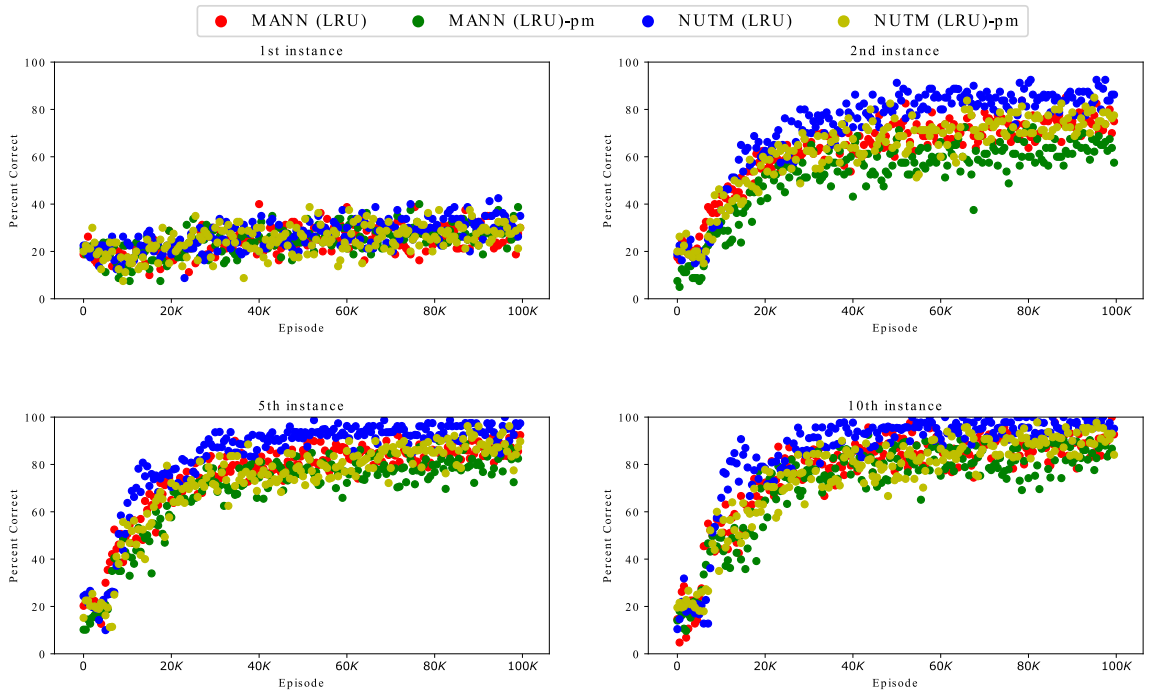Testing accuracy through time is listed below,



Figure E.17: Testing accuracy during training (five random classes/episode, one-hot vector labels, of length 50).

| Model | Persistent memory[4] | 5 classes | | | 10 classes | | |
|---|---|---|---|---|---|---|---|
| | | $2^{nd}$ | $3^{rd}$ | $5^{th}$ | $2^{nd}$ | $3^{rd}$ | $5^{th}$ |
| MANN (LRUA)* | No | 82.8 | 91.0 | 94.9 | - | - | - |
| MANN (LRUA) | No | 82.3 | 88.7 | 92.3 | 52.7 | 60.6 | 64.7 |
| NUTM (LRUA) | No | **85.7** | **91.3** | **95.5** | **68.0** | **78.1** | **82.8** |
| Human* | Yes | 57.3 | 70.1 | 81.4 | - | - | - |
| MANN (LRUA)* | Yes | $\approx 58.0$ | - | $\approx 75.0$ | $\approx 60.0$ | - | $\approx 80.0$ |
| MANN (LRUA) | Yes | 66.2 | 73.4 | 81.0 | 51.3 | 59.2 | 63.3 |
| NUTM (LRUA) | Yes | **77.8** | **85.8** | **89.8** | **69.0** | **77.9** | **82.7** |

Table E.7: Test-set classification accuracy (%) on the Omniglot dataset after 100,000 episodes of training. * denotes available results from Santoro et al., (2016) (some are estimated from plotted figures).
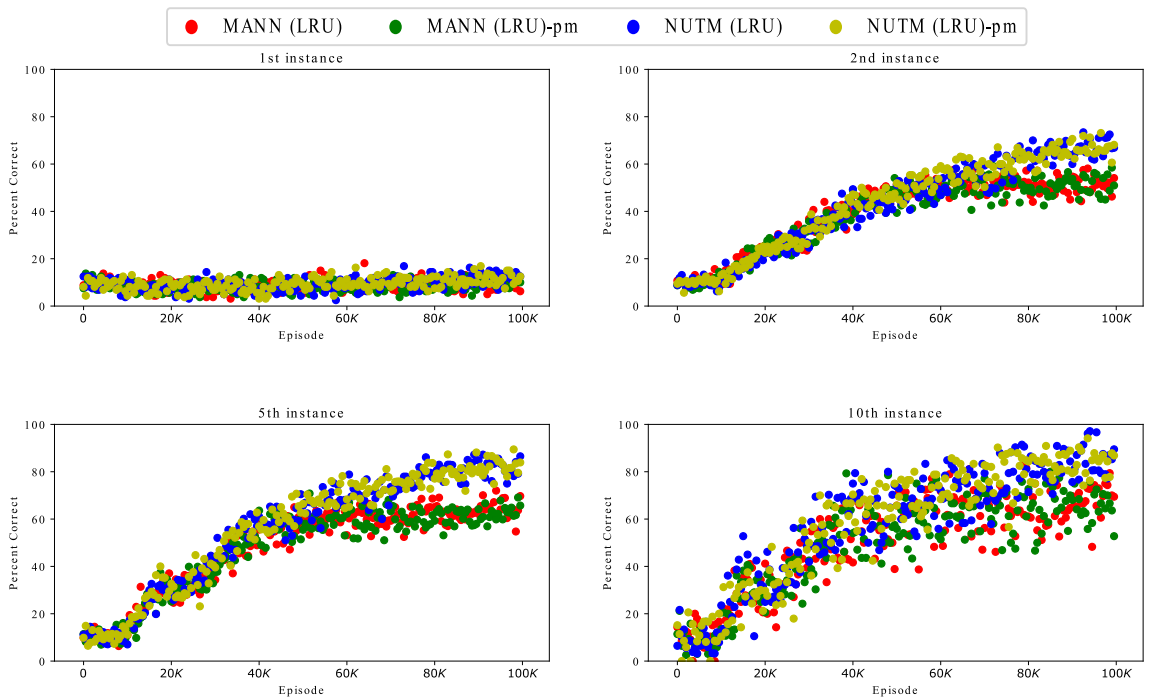


Figure E.18: Testing accuracy during training (ten random classes/episode, one-hot vector labels, of length 75).

Final testing accuracy is listed as follows,

It should be noted that our goal was not to achieve state of the art performance on this dataset. It was to exhibit the benefit of NSM to MANN. Compared to current methods, the MANN and NUTM used in our experiments do not use CNN to extract visual features, thus achieve lower accuracy.

---

[4]If the memory is not artificially erased between episodes, it is called persistent. This mode is hard for the case of 5 classes (Santoro et al., 2016)

## E.6    Details on bAbI Task

We train the models using RMSprop optimiser with fixed learning rate of $10^{-4}$ and momentum of 0.9. The batch size is 32 and we adopt layer normalisation (Lei Ba et al., 2016) to DNC's layers. Following common practices (Franke et al., 2018), we also remove temporal linkage for faster training. The details of hyper-parameters are listed in Table E.8. Full NUTM ($p = 4$) results are reported in Table E.9.

| #Head | Controller Size | $N$ | $M$ | $p$ | $\mathbf{M}_p.K$ Size | #Parameters |
|:-----:|:---------------:|:---:|:---:|:---:|:---------------------:|:-----------:|
| 4 | 172 | 196 | 64 | 4 | 4 | 794,773 |
| 4 | 200 | 196 | 64 | 2 | 2 | 934,787 |

Table E.8: NUTM hyper-parameters for bAbI.

| Task | bAbI Best Results | bAbI Mean Results |
|:-----|:-----------------:|:-----------------:|
| 1: 1 supporting fact | 0.0 | $0.0 \pm 0.0$ |
| 2: 2 supporting facts | 0.2 | $0.6 \pm 0.3$ |
| 3: 3 supporting facts | 4.0 | $7.6 \pm 3.9$ |
| 4: 2 argument relations | 0.0 | $0.0 \pm 0.0$ |
| 5: 3 argument relations | 0.4 | $1.0 \pm 0.4$ |
| 6: yes/no questions | 0.0 | $0.0 \pm 0.0$ |
| 7: counting | 1.9 | $1.5 \pm 0.8$ |
| 8: lists/sets | 0.6 | $0.3 \pm 0.2$ |
| 9: simple negation | 0.0 | $0.0 \pm 0.0$ |
| 10: indefinite knowledge | 0.1 | $0.1 \pm 0.0$ |
| 11: basic coreference | 0.0 | $0.0 \pm 0.0$ |
| 12: conjunction | 0.0 | $0.0 \pm 0.0$ |
| 13: compound coreference | 0.1 | $0.0 \pm 0.0$ |
| 14: time reasoning | 0.3 | $1.6 \pm 2.2$ |
| 15: basic deduction | 0.0 | $2.6 \pm 8.3$ |
| 16: basic induction | 49.3 | $52.0 \pm 1.7$ |
| 17: positional reasoning | 4.7 | $18.4 \pm 12.7$ |
| 18: size reasoning | 0.4 | $1.6 \pm 1.1$ |
| 19: path finding | 4.3 | $23.7 \pm 32.2$ |
| 20: agent's motivation | 0.0 | $0.0 \pm 0.0$ |
| Mean Error (%) | 3.3 | $5.6 \pm 1.9$ |
| Failed (Err. >5%) | 1 | $3 \pm 1.2$ |

Table E.9: NUTM ($p = 4$) bAbI best and mean errors (%).

## E.7 Others

If we deliberately set the key dimension equal to the number of programs, we can even place an orthogonal basis constraint on the key space of NSM by minimising the following loss,

$$l_{p_2} = \left\| \mathbf{M}_p.K\mathbf{M}_p.K^T - \mathbf{I} \right\| \tag{8.8}$$

where $\mathbf{M}_p.K$ and $\mathbf{I}$ denote the key part in NSM and the identity matrix, respectively.

For all tasks, $\eta_t$ is fixed to 0.1, reducing with decay rate of 0.9.

# Bibliography

Daniel J Amit. *Modeling brain function: The world of attractor neural networks.* Cambridge university press, 1992.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1545–1554, 2016a. doi: 10.18653/v1/N16-1181. URL https://www.aclweb.org/anthology/N16-1181.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016b.

Jennifer Hill Andrew Gelman. *Data Analysis Using Regression and Multilevel/Hierarchical Models.* Cambridge University Press, 2006.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.

Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 233–242. JMLR. org, 2017.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, pages 4331–4339, 2016.

Athanassia Bacharoglou. Approximation of probability distributions by convex mixtures of gaussian measures. *Proceedings of the American Mathematical Society*, 138(7):2619–2628, 2010.

Alan Baddeley. Working memory. *Science*, 255(5044):556–559, 1992.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *Proceedings of the International Conference on Learning Representations*, 2015.

Jacek M Bajor and Thomas A Lasko. Predicting medications from diagnostic codes with recurrent neural networks. In *International Conference on Learning Representations*, 2017.

Eric B Baum, John Moody, and Frank Wilczek. Internal representations for associative memory. *Biological Cybernetics*, 59(4-5):217–228, 1988.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5 (2):157–166, 1994.

Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM, 1998.

Hal Blumenfeld. *Neuroanatomy through Clinical Cases*. Oxford University Press, 2010.

Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

Jörg Bornschein, Andriy Mnih, Daniel Zoran, and Danilo Jimenez Rezende. Variational memory addressing in generative models. In *Advances in Neural Information Processing Systems*, pages 3923–3932, 2017.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of The SIGNLL Conference on Computational Natural Language Learning*, pages 10–21, 2016.

Denny Britz, Melody Guan, and Minh-Thang Luong. Efficient attention using a fixed-size memory representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 392–400, 2017.

Mike Casey. The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural computation*, 8 (6):1135–1178, 1996.

Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 77–87, 2017.

Boxing Chen and Colin Cherry. A systematic comparison of smoothing techniques for sentence-level bleu. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 362–367, 2014.

Hongshen Chen, Zhaochun Ren, Jiliang Tang, Yihong Eric Zhao, and Dawei Yin. Hierarchical variational memory network for dialogue generation. In *Proceedings of the World Wide Web Conference on World Wide Web*, pages 1653–1662. International World Wide Web Conferences Steering Committee, 2018.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014a.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014b. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1179.

Edward Choi, Mohammad Taha Bahadori, and Jimeng Sun. Doctor AI: Predicting Clinical Events via Recurrent Neural Networks. *arXiv preprint arXiv:1511.05942*, 2015.

Edward Choi, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter Stewart. RETAIN: An Interpretable Predictive Model for Healthcare using Reverse Time Attention Mechanism. In *Advances in Neural Information Processing Systems*, pages 3504–3512, 2016.

J. S. Chung, A. Senior, O. Vinyals, and A. Zisserman. Lip reading sentences in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems*, pages 2980–2988, 2015.

Dan C Cireşan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In *International Conference on Medical Image Computing and Computer-assisted Intervention*, pages 411–418. Springer, 2013.

Patricia Cohen, Stephen G West, and Leona S Aiken. *Applied multiple regression/correlation analysis for the behavioral sciences*. Psychology Press, 2014.

Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for natural language processing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, 2016.

Robert Csordas and Juergen Schmidhuber. Improving differentiable neural computers through memory masking, de-allocation, and link distribution sharpness control. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyGEM3C9KQ.

Clayton E Curtis and Mark D'Esposito. Persistent activity in the prefrontal cortex during working memory. *Trends in cognitive sciences*, 7(9):415–423, 2003.

Ivo Danihelka, Greg Wayne, Benigno Uria, Nal Kalchbrenner, and Alex Graves. Associative long short-term memory. In *International Conference on Machine Learning*, pages 1986–1994, 2016.

Nat Dilokthanakul, Pedro AM Mediano, Marta Garnelo, Matthew CH Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*, 2016.

J-L Durrieu, J-Ph Thiran, and Finnian Kelly. Lower and upper bounds for approximation of the kullback-leibler divergence between gaussian mixture models. In *IEEE International Conference on Acoustics, Speech and Signal Processing.*, 2012.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Gabriel Forgues, Joelle Pineau, Jean-Marie Larchevêque, and Réal Tremblay. Bootstrapping dialog systems with word embeddings. In *Nips, Modern Machine Learning and Natural Language Processing Workshop*, volume 2, 2014.

Jörg Franke, Jan Niehues, and Alex Waibel. Robust and scalable differentiable neural computer for question answering. In *Proceedings of the Workshop on Machine Reading for Question Answering*, pages 47–59. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/W18-2606.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

Surya Ganguli, Dongsung Huh, and Haim Sompolinsky. Memory traces in dynamical systems. *Proceedings of the National Academy of Sciences*, 105(48):18970–18975, 2008.

Jiyang Gao, Runzhou Ge, Kan Chen, and Ram Nevatia. Motion-appearance co-memory networks for video question answering. *arXiv preprint arXiv:1803.10906*, 2018.

Mevlana Gemici, Chia-Chun Hung, Adam Santoro, Greg Wayne, Shakir Mohamed, Danilo J Rezende, David Amos, and Timothy Lillicrap. Generative temporal models with memory. *arXiv preprint arXiv:1702.04649*, 2017.

C Lee Giles, Clifford B Miller, Dong Chen, Hsing-Hen Chen, Guo-Zheng Sun, and Yee-Chun Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

Patricia S Goldman-Rakic. Cellular basis of working memory. *Neuron*, 14(3):477–485, 1995.

Alejandro González, Gabriel Villalonga, Jiaolong Xu, David Vázquez, Jaume Amores, and Antonio M López. Multiview random forest of local experts combining rgb and lidar data for pedestrian detection. In *Intelligent Vehicles Symposium (IV), 2015 IEEE*, pages 356–361. IEEE, 2015.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

Edouard Grave, Moustapha M Cisse, and Armand Joulin. Unbounded cache model for online language modeling with open vocabulary. In *Advances in Neural Information Processing Systems*, pages 6042–6052, 2017a.

Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. In *International Conference on Learning Representations*, 2017b.

A. Graves, G. Wayne, and I. Danihelka. Neural Turing Machines. *ArXiv e-prints*, October 2014.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6): 602–610, 2005.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, pages 1828–1836, 2015.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

Caglar Gulcehre, Sarath Chandar, and Yoshua Bengio. Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv:1701.08718*, 2017.

Sadid A Hasan, Siyuan Zhao, Vivek V Datla, Joey Liu, Kathy Lee, Ashequl Qadir, Aaditya Prakash, and Oladimeji Farri. Clinical question answering using key-value memory networks and knowledge graph. In *TREC*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Donald Olding Hebb. *The organization of behavior: a neuropsychological theory.* Science Editions, 1962.

John R Hershey and Peder A Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *IEEE International Conference on Acoustics, Speech and Signal Processing.*, 2007.

Geoffrey E Hinton and David C Plaut. Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the Cognitive Science Society*, pages 177–186, 1987.

Geoffrey E Hinton et al. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA, 1986.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8): 2554–2558, 1982.

Francis L Huang. Multilevel modeling and ordinary least squares regression: how comparable are they? *The Journal of Experimental Education*, 86(2):265–281, 2018.

Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1965–1972. International Joint Conference on Artificial Intelligence, 2017.

Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. MIMIC-III, a freely accessible critical care database. *Scientific data*, 3, 2016.

Samuel Johnson, J Marro, and Joaquín J Torres. Robust short-term memory without synaptic learning. *PloS one*, 8(1):e50276, 2013.

Michael I Jordan. Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier, 1997.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in Neural Information Processing Systems*, pages 190–198, 2015.

Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, 2013.

Pentti Kanerva. *Sparse distributed memory*. MIT press, 1988.

James D Keeler. Comparison between kanerva's sdm and hopfield-type neural networks. *Cognitive Science*, 12(3):299–329, 1988.

Asjad Khan, Hung Le, Kien Do, Truyen Tran, Aditya Ghose, Hoa Dam, and Renuka Sindhgatta. Memory-augmented neural networks for predictive process analytics. *arXiv preprint arXiv:1802.00938*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations*, 2014.

Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

Iuliia Kotseruba and John K Tsotsos. 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review*, pages 1–78, 2018.

David Krueger, Nicolas Ballas, Stanislaw Jastrzebski, Devansh Arpit, Maxinder S Kanwal, Tegan Maharaj, Emmanuel Bengio, Asja Fischer, and Aaron Courville. Deep nets don't learn via memorization. 2017.

Dharshan Kumaran, Demis Hassabis, and James L McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in cognitive sciences*, 20(7):512–534, 2016.

Hung Le, Truyen Tran, Thin Nguyen, and Svetha Venkatesh. Variational memory encoder-decoder. In *Advances in Neural Information Processing Systems*, pages 1508–1518, 2018a.

Hung Le, Truyen Tran, and Svetha Venkatesh. Dual memory neural computer for asynchronous two-view sequential learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery; Data Mining*, KDD '18, pages 1637–1645, New York, NY, USA, 2018b. ACM. ISBN 978-1-4503-5552-0. doi: 10.1145/3219819.3219981. URL http://doi.acm.org/10.1145/3219819.3219981.

Hung Le, Truyen Tran, and Svetha Venkatesh. Dual control memory augmented neural networks for treatment recommendations. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 273–284. Springer, 2018c.

Hung Le, Truyen Tran, and Svetha Venkatesh. Learning to remember more with less memorization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=r1xlvi0qYm.

Hung Le, Truyen Tran, and Svetha Venkatesh. Neural stored-program memory. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkxxA24FDr.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, pages 2177–2185, 2014.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 110–119, 2016.

Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzel. Learning to Diagnose with LSTM Recurrent Neural Networks. In *International Conference on Learning Representations*, 2016.

Pierre Lison and Serge Bibauw. Not all dialogues are created equal: Instance weighting for neural conversational models. In *Proceedings of the Annual SIGdial Meeting on Discourse and Dialogue*, pages 384–394, 2017.

Robert H Logie. *Visuo-spatial working memory.* Psychology Press, 2014.

David Lopez-Paz et al. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017.

Oscar Luaces, Jorge Díez, José Barranquero, Juan José del Coz, and Antonio Bahamonde. Binary relevance efficacy for multilabel classification. *Progress in Artificial Intelligence*, 1(4):303–313, 2012.

Chao Ma, Chunhua Shen, Anthony Dick, and Anton van den Hengel. Visual question answering with memory-augmented networks. *arXiv preprint arXiv:1707.04968*, 2017a.

Fenglong Ma, Radha Chitta, Jing Zhou, Quanzeng You, Tong Sun, and Jing Gao. Dipole: Diagnosis prediction in healthcare via attention-based bidirectional recurrent neural networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1903–1911. ACM, 2017b.

Ying Ma and Jose Principe. A taxonomy for neural memory networks. *arXiv preprint arXiv:1805.00327*, 2018.

Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

Wolfgang Maass. Liquid state machines: motivation, theory, and applications. In *Computability in context: computation and logic in the real world*, pages 275–296. World Scientific, 2011.

Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

David Marr and W Thomas Thach. A theory of cerebellar cortex. In *From the Retina to the Neocortex*, pages 11–50. Springer, 1991.

Vladimir Maz'ya and Gunther Schmidt. On approximate approximations using gaussian kernels. *IMA Journal of Numerical Analysis*, 16(1):13–29, 1996.

James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1400–1409, 2016.

John Miller and Moritz Hardt. When recurrent models don't need to be recurrent. *arXiv preprint arXiv:1805.10369*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Louis-Philippe Morency, Rada Mihalcea, and Payal Doshi. Towards multimodal sentiment analysis: Harvesting opinions from the web. In *Proceedings of the 13th international conference on multimodal interfaces*, pages 169–176. ACM, 2011.

Michael C Mozer and Sreerupa Das. A connectionist symbol manipulator that discovers the structure of context-free languages. In *Advances in Neural Information Processing Systems*, pages 863–870, 1993.

Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2554–2563. JMLR. org, 2017.

Eric Nalisnick, Lars Hertel, and Padhraic Smyth. Approximate inference for deep latent gaussian mixtures. In *NIPS Workshop on Bayesian Deep Learning*, volume 2, 2016.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of the SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, 2016.

Phuoc Nguyen, Truyen Tran, Nilmini Wickramasinghe, and Svetha Venkatesh. Deepr: A Convolutional Net for Medical Records. *Journal of Biomedical and Health Informatics*, 21(1), 2017.

Liqiang Nie, Luming Zhang, Yi Yang, Meng Wang, Richang Hong, and Tat-Seng Chua. Beyond doctors: future health prediction from multimedia and multimodal observations. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 591–600. ACM, 2015.

Kamal Nigam and Rayid Ghani. Analyzing the effectiveness and applicability of co-training. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 86–93. ACM, 2000.

Christian W Omlin and C Lee Giles. Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM (JACM)*, 43(6):937–972, 1996.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Predicting healthcare trajectories from medical records: A deep learning approach. *Journal of Biomedical Informatics*, 69:218–229, May 2017.

Tony A Plate. Holographic reduced representations. *IEEE Transactions on Neural networks*, 6(3):623–641, 1995.

Kim Plunkett and Chris Sinha. Connectionism and developmental theory. *British Journal of Developmental Psychology*, 10(3):209–254, 1992.

Soujanya Poria, Erik Cambria, and Alexander Gelbukh. Deep convolutional neural network textual features and multiple kernel learning for utterance-level multimodal sentiment analysis. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 2539–2544, 2015.

Soujanya Poria, Erik Cambria, Devamanyu Hazarika, Navonil Majumder, Amir Zadeh, and Louis-Philippe Morency. Context-dependent sentiment analysis in user-generated videos. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 873–883, 2017.

Aaditya Prakash, Siyuan Zhao, Sadid A Hasan, Vivek V Datla, Kathy Lee, Ashequl Qadir, Joey Liu, and Oladimeji Farri. Condensed memory networks for clinical diagnostic inferencing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3274–3280, 2017.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. *arXiv preprint arXiv:1703.01988*, 2017.

N. Quadrianto, A.J. Smola, T.S. Caetano, and Q.V. Le. Estimating labels from label proportions. *The Journal of Machine Learning Research*, 10:2349–2374, 2009.

Chao Qui, Bo Huang, Guocheng Niu, Daren Li, Daxiang Dong, Wei He, Dianhai Yu, and Hua Wu. A new method of region embedding for text classification. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BkSDMA36Z.

Jack Rae, Jonathan J Hunt, Ivo Danihelka, Timothy Harley, Andrew W Senior, Gregory Wayne, Alex Graves, and Tim Lillicrap. Scaling memory-augmented neural networks with sparse reads and writes. In *Advances in Neural Information Processing Systems*, pages 3621–3629, 2016.

Shyam Sundar Rajagopalan, Louis-Philippe Morency, Tadas Baltrusaitis, and Roland Goecke. Extending long short-term memory for multi-view structured learning. In *European Conference on Computer Vision*, pages 338–353. Springer, 2016.

Alain Rakotomamonjy, Francis Bach, Stéphane Canu, and Yves Grandvalet. More efficiency in multiple kernel learning. In *Proceedings of the 24th international conference on Machine learning*, pages 775–782. ACM, 2007.

Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333–359, 2011.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic back-propagation and approximate inference in deep generative models. In *Proceedings of the International Conference on International Conference on Machine Learning*, pages II–1278. JMLR. org, 2014.

Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *Journal of machine learning research*, 5(Jan):101–141, 2004.

Timothy T Rogers and James L McClelland. *Semantic cognition: A parallel distributed processing approach*. MIT press, 2004.

Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Rodrigo Salgado, Francisco Bellas, Pilar Caamaño, Borja Santos-Diez, and Richard J Duro. A procedural long term memory for cognitive robotics. In *2012 IEEE Conference on Evolving and Adaptive Intelligent Systems*, pages 57–62. IEEE, 2012.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016.

Andrew M Saxe, James L McClelland, and Surya Ganguli. A mathematical theory of semantic development in deep neural networks. *Proceedings of the National Academy of Sciences*, 116(23):11537–11546, 2019.

Imanol Schlag and Jürgen Schmidhuber. Learning to reason with third order tensor products. In *Advances in Neural Information Processing Systems*, pages 9981–9993, 2018.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. Neural Speed Reading via Skim-RNN. In *International Conference on Learning Representations*, 2018.

Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron C Courville, and Yoshua Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3295–3301, 2017.

Xiaoyu Shen, Hui Su, Yanran Li, Wenjie Li, Shuzi Niu, Yang Zhao, Akiko Aizawa, and Guoping Long. A conditional variational framework for dialog generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 504–509, 2017.

Rui Shu, James Brofos, Frank Zhang, Hung Hai Bui, Mohammad Ghavamzadeh, and Mykel Kochenderfer. Stochastic video prediction with conditional density estimation. In *ECCV Workshop on Action and Anticipation for Visual Learning*, volume 2, 2016.

Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150, 1995.

Michael Sipser et al. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216, 1990.

Yale Song, Louis-Philippe Morency, and Randall Davis. Multi-view latent variable discriminative models for action recognition. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2120–2127. IEEE, 2012.

Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, pages 2377–2385, 2015.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in Neural Information Processing Systems*, pages 2440–2448, 2015.

David Sussillo. Neural circuits as computational dynamical systems. *Current opinion in neurobiology*, 25:156–163, 2014.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *International Conference on Machine Learning*, 2014a. URL http://arxiv.org/abs/1409.3215.

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014b.

Jie Tan, Matthew Ung, Chao Cheng, and Casey S Greene. Unsupervised feature construction and knowledge extraction from genome-wide assays of breast cancer with denoising autoencoders. In *Pacific Symposium on Biocomputing Co-Chairs*, pages 132–143. World Scientific, 2014.

Sebastian Thrun. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer, 1998.

Peter Tiňo, Bill G Horne, C Lee Giles, and Pete C Collingwood. Finite state machines and recurrent neural networks–automata and dynamical systems approaches. In *Neural networks and pattern recognition*, pages 171–219. Elsevier, 1998.

Trieu H Trinh, Andrew M Dai, Thang Luong, and Quoc V Le. Learning longer-term dependencies in rnns with auxiliary losses. In *Proceedings of the 35 th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80*, 2018.

Endel Tulving et al. Episodic and semantic memory. *Organization of memory*, 1: 381–403, 1972.

Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. In *Proceedings of the London Mathematical Society*, 1936.

Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, page 125, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

John von Neumann. First draft of a report on the edvac. *IEEE Ann. Hist. Comput.*, 15(4):27–75, October 1993. ISSN 1058-6180. doi: 10.1109/85.238389. URL https://doi.org/10.1109/85.238389.

Liwei Wang, Alexander Schwing, and Svetlana Lazebnik. Diverse and accurate image description using a variational auto-encoder with an additive gaussian encoding space. In *Advances in Neural Information Processing Systems*, pages 5756–5766, 2017.

Mingxuan Wang, Zhengdong Lu, Hang Li, and Qun Liu. Memory-enhanced decoder for neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 278–286, 2016.

Tsung-Hsien Wen, Yishu Miao, Phil Blunsom, and Steve Young. Latent intention dialogue models. In *Proceedings of the International Conference on Machine Learning*, pages 3732–3741, 2017.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

Faustine Williams. The role of the electronic medical record (emr) in care delivery development in developing countries: a systematic review. *Informatics in Primary Care Journal*, 2008.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 4880–4888, 2016.

Chang Xu, Dacheng Tao, and Chao Xu. A survey on multi-view learning. *arXiv preprint arXiv:1304.5634*, 2013.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

Dani Yogatama, Chris Dyer, Wang Ling, and Phil Blunsom. Generative and discriminative text classification with recurrent neural networks. *arXiv preprint arXiv:1703.01898*, 2017.

Dani Yogatama, Yishu Miao, Gabor Melis, Wang Ling, Adhiguna Kuncoro, Chris Dyer, and Phil Blunsom. Memory architectures in recurrent neural network language models. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SkFqf0lAZ.

Adams Wei Yu, Hongrae Lee, and Quoc Le. Learning to skim text. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1880–1890, 2017.

Keyi Yu, Yang Liu, Alexander G Schwing, and Jian Peng. Fast and accurate text classification: Skimming, rereading and early stopping. *ICLR Workshop Track*, 2018.

Amir Zadeh, Rowan Zellers, Eli Pincus, and Louis-Philippe Morency. Multimodal sentiment intensity analysis in videos: Facial gestures and verbal messages. *IEEE Intelligent Systems*, 31(6):82–88, 2016.

Amir Zadeh, Minghai Chen, Soujanya Poria, Erik Cambria, and Louis-Philippe Morency. Tensor fusion network for multimodal sentiment analysis. *arXiv preprint arXiv:1707.07250*, 2017.

Amir Zadeh, Paul Pu Liang, Navonil Mazumder, Soujanya Poria, Erik Cambria, and Louis-Philippe Morency. Memory fusion network for multi-view sequential learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3987–3995. JMLR. org, 2017.

Biao Zhang, Deyi Xiong, Hong Duan, Min Zhang, et al. Variational neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 521–530, 2016a.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016b.

Yutao Zhang, Robert Chen, Jie Tang, Walter F Stewart, and Jimeng Sun. Leap: Learning to prescribe effective and safe treatment combinations for multimorbidity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1315–1324. ACM, 2017.

Tiancheng Zhao, Ran Zhao, and Maxine Eskenazi. Learning discourse-level diversity for neural dialog models using conditional variational autoencoders. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 654–664, 2017.

Jozef Zurada. End effector target position learning using feedforward with error back-propagation and recurrent neural networks. In *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, volume 4, pages 2633–2638. IEEE, 1994.

———————————————

This Agreement between Mr. Hung Le ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

| | |
|---|---|
| License Number | 4674490942813 |
| License date | Sep 22, 2019 |
| Licensed Content Publisher | Springer Nature |
| Licensed Content Publication | Springer eBook |
| Licensed Content Title | Dual Control Memory Augmented Neural Networks for Treatment Recommendations |
| Licensed Content Author | Hung Le, Truyen Tran, Svetha Venkatesh |
| Licensed Content Date | Jan 1, 2018 |
| Type of Use | Thesis/Dissertation |
| Requestor type | academic/university or research institute |
| Format | print and electronic |
| Portion | full article/chapter |
| Will you be translating? | no |
| Circulation/distribution | 1 - 29 |
| Author of this Springer Nature content | yes |
| Title | Memory and attention in deep learning |
| Institution name | Deakin University |
| Expected presentation date | Jan 2020 |
| Requestor Location | Mr. Hung Le 4/1A Villamanta Street, Geelong West |
| | Geelong, Vic 3218 Australia Attn: Mr. Hung Le |
| Total | 0.00 USD |

Terms and Conditions